SIEMENS

COB1 (BS2000)
COBOL Compiler

Reference Manual
Part 2

Edition August 1986 (Software Product COB1 V2.3A)

Order No. U996-J-Z55-6-7600 Printed in the Federal Republic of Germany 2950 AG 11860.8 (3690)	Order No.: U996-J-Z55-6-7600	A ring binder for this manual is available at a cost of DM 4,20 Order number: U1225-J-Z18-1

COB1 (BS2000)
COBOL Compiler

Reference Manual
Part 2

Edition August 1986 (Software Product COB1 V2.3A)

Order No. U996-J-Z55-6-7600
Printed in the Federal Republic of Germany
2950 AG 11860.8 (3690)

COB1 (BS2000)
COBOL Compiler

Reference Manual
Part 2

Edition August 1986 (Software Product COB1 V2.3A)

Order No. U996-J-Z55-6-7600
Printed in the Federal Republic of Germany
2950 AG 11860.8 (3690)

Siemens Aktiengesellschaft

PREFACE

This manual describes the COBOL language for the Siemens COBOL compiler COB1, Version 2.3A, BS2000 Operating System.

The manual contains all of those language elements which might possibly occur in the creation of COBOL programs, divided according to function, format, rules, programming considerations and examples.

It is intended as a guide to the writing of COBOL programs and as a complement to training manuals. Knowledge of COBOL is a prerequisite.

The structure follows the language modules defined in the Standard for COBOL. Extensions to and deviations from Standard 74 are marked with colored shading as follows:

Gray shading = Siemens COB1 extensions
Orange shading = Language elements to be avoided wherever possible since they will no longer be supported by future COBOL standards.

The concepts and expressions used in this manual are defined, in alphabetical order, in the "Glossary".

For further prefatory information consult chapter 1, "Introduction".

Changes made to the previous manual are summarized in the "List of Amendments".

Literature references in the text are given in short-title format. The complete title of each publication cited is listed in the "Literature" section.

To help us continue to improve our publications, please send us your comments, requests and suggestions using the pink reply forms provided.

Editorial Department K D ST QM 2

ACKNOWLEDGMENT

COBOL, the programming language described in this manual, is based on the language laid down in the standard document "American National Standard Programming Language COBOL, X.3.23-1974". In recognition of the efforts made to develop and standardize COBOL it is customary to precede a description of COBOL with the following text as it appears in the original publication:

Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas from this report as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce the following acknowledgment paragraphs in their entirety as part of the preface to any such publication. Any organization using a short passage from this document, such as in a book review, is requested to mention COBOL in acknowledgment of the source, but need not quote the acknowledgment.

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL Programming Language Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted material used herein

FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

LIST OF AMENDMENTS

The following amendments have been made since publication of the
Corrections February 1986 (COB1 Version 2.2A)
and incorporated in the
Edition August 1986 (COB1 Version 2.3A)

1. All references to BS1000 have been removed.

2. The following language elements are no longer described since they will not be supported by future versions of the compiler:

Statements	ALTER CALL USING procedure-name *DEBUG ENTER EXAMINE EXHIBIT GO TO to/from USE procedures ON TRACE TRANSFORM
Phrases	ALTERNATE in RESERVE clause COBRUN QUOTE1 COMP-4 CURRENT DATE/TODAYS DATE FOR MULTIPLE REEL OTHERWISE POSITIONING
Clauses	FILE-LIMIT PROCESSING MODE SYMBOLIC KEY/NOMINAL KEY TRACK-AREA VALUE OF
Paragraphs	DATE-COMPILED REMARKS

3. New language elements:

   FILE STATUS...data-name-2
   INITIALIZE
   START KEY LESS
   SYSLSTnn support

4. Other important changes:

Page	Item changed	new	modi-fied	dis-carded
1-6	Character set: lower case	X		
1-41	RETURN-CODE: gray shading	X		
1-50	Numeric floating-point literals: format		X	
2-2	Headings of program divisions		X	
2-15	SPECIAL-NAMES: format		X	
2-32	77-level description entry		X	
2-42	Data description entries for group items		X	
2-65	Example 22		X	
2-91	COMPUTATIONAL: Rule e)			X
2-91	Length of paragraph/section names		X	
2-100	Relation Condition: table		X	
2-103	Note 4			X
2-125	DAY-OF-WEEK: gray shading			X
2-127	ADD statement: format 2		X	
2-133	DISPLAY: Programming consideration 6	X		
2-142	GO TO statement, format 3: orange shading	X		
3-12	ORGANIZATION clause: format		X	
3-20	File description entry: rules		X	
3-32	RECORD clause: format 1 + rules		X	
3-35	RECORDING MODE: rules		X	
3-46	READ statement: rule 5		X	
4-15	ORGANIZATION clause: format		X	
4-23	File description entry: rules		X	
5-6	Message 94		X	
5-14	ORGANIZATION clause: format		X	
5-22	File description entry: rules		X	
7-6	CALL statement: rule 4		X	
7-11	EXIT PROGRAM: rules		X	
8-10	Control fields: rule 8			X
	Literature		X	

CONTENTS

Contents

Contents

Contents

Contents

TABLES

TABLES AND FIGURES IN CHAPTER 1

Tables

Figures

TABLES AND FEATURES IN CHAPTER 2

Tables

Figures

TABLES IN CHAPTER 3

TABLES IN CHAPTER 4

TABLES IN CHAPTER 5

TABLES IN CHAPTER 6

Tables

Figures

TABLES IN CHAPTER 8

4 RELATIVE FILE ORGANIZATION

4.1 FILE CONCEPTS

A file is a collection of data records that can be transferred to, or read from, a volume. The user defines the organization of the file as well as the mode and order of file processing.

The organization of a file describes its logical structure. There are sequential, indexed, relative, and direct types of file organization. The file organization which is defined at the time a file is constructed cannot be changed later on. This same rule applies to the record size of a particular file, as defined in the file and record description entries.

4.1.1 Relative Organization

When using relative file organization, the location of each data record in a relative file is determined by means of a relative record number, i.e. an integer value greater than zero, which specifies the position of that record within the logical sequence of the file. The record number is predefined by the user in a relative key field. The file may be seen as a serial sequence of areas, each of which contains one logical data record. Each of these areas is identified by a relative record number. Storage and retrieval of the data records is accomplished on the basis of that number. For example, the tenth data record is referenced through the relative record number 10 and is located in the tenth record area, whether or not data records have been written in any of the record areas 1 to 9. Relative file organization is permitted for disk-storage files only.

4.1.2 Sequential Access to Data Records in a Relative File

Data records in a relative file may be sequentially created, read, and updated.

A relative file may be created sequentially; in this case, the RELATIVE KEY need not be specified, as the records are written to the output file in physically consecutive order. For preformatting see the "COB1 User's Guide" [7].

When sequentially reading data records from a relative file, the records are processed in the order of their relative record numbers, i.e. a READ statement will supply the next logical record of the file. It is not necessary for the next logical record being read to follow the preceding record in a physical sense, if records with intervening relative record numbers are not logically present on the file but do exist only physically as auxiliary records.

The first data record being read is either the first record made available in the file, or it is a record whose position was specified in a START statement; in this case, the RELATIVE KEY clause will be required.

If the RELATIVE KEY is specified for a relative file, execution of a READ or WRITE statement causes the RELATIVE KEY item to be set to the relative record number of the data record which is made available.

An already existing relative file may be updated with the aid of READ, REWRITE, or DELETE statements. The last record read may be updated and may then be rewritten to its original location in the file, or it may be logically deleted.

4.1.3 Random Access to Data Records of a Relative File

Data records of a relative file may be randomly created, read, updated, and added.

With this access method, the RELATIVE KEY clause is always required. Before each execution of an input statement or output statement, the data item indicated in the RELATIVE KEY clause must be provided with the value of the required relative record number.

A relative file may be created randomly; in this case, the record placed on the file occupies the record area of the relative record number as specified in the RELATIVE KEY field. All unassigned record areas are preformatted by the runtime system with auxiliary records (marked with HIGH-VALUES in their first byte).

When a relative file is read randomly, the record being retrieved is the one whose relative record number is supplied in the data item of the RELATIVE KEY clause associated with that file.

An already existing relative file may be updated with the aid of REWRITE or DELETE statements. An explicit random READ statement may optionally be issued prior to the execution of a REWRITE or DELETE statement for the data record to be updated, as these statements will modify the record whose relative record number is supplied in the data item of the RELATIVE KEY clause.

4.1.4 Dynamic Access to Data Records in a Relative File

Switching from sequential to random access mode.

4.1.5 Valid Access Methods

File organization and possible actions	Allowable access methods (X) Sequential	Random	Extended	Dynamic
Relative Files				
Record creation	X	X		X
Record retrieval	X	X		X
Record update	X	X		X
Record addition		X		X

Table 4-1 Valid access methods

4.1.6 I-O Status

The I-O status is a value which is transferred to a two-character data item for the purpose of indicating to the COBOL program the status of an input/output operation. The takes place during execution of a CLOSE, DELETE, OPEN, READ, REWRITE, START or WRITE statement and prior to execution of any associated imperative statement and any corresponding USE AFTER STANDARD EXCEPTION procedure. The I-O status value is transferred only if the FILE STATUS clause is specified in the FILE-CONTROL paragraph (see FILE STATUS clause).

At the termination of an I-O operation the values of the I-O status have the following meaning:

1. Execution successful
   The I-O statement was processed normally.

2. Execution unsuccessful: AT END condition
   A READ statement terminated abnormally due to an AT END condition.

3. Execution unsuccessful: invalid key condition
   The I-O statement terminated abnormally due to an invalid key condition.

4. Execution unsuccessful: unrecoverable error
   The I-O statement terminated abnormally due to the occurrence of an unrecoverable error precluding further processing of the file. An unrecoverable error cannot be rectified within the COBOL program.

5. Execution unsuccessful: logical error
   An I-O statement terminated abnormally due to an illegal sequence of I-O operations for this file.

6. Conflict between the permanent characteristics of a file and those described in the program.
   The I-O statement terminated abnormally because the permanent file characteristics are incompatible with those given in the COBOL program.

7. System error
   The I-O statement terminated abnormally due to a system error.

The table below shows the values of the I-O status and their meanings:

I-O status	Meaning
	Execution successful
00	- The I-O statement terminated normally. No further information regarding the I-O operation is available.
04	- A READ statement terminated normally. However, the length of the records read exceeds the limits defined for this file.
	Execution unsuccessful: AT END condition
10	- An attempt was made to execute a READ statement. However, the next text logical was nonexistent as the end-of-file was encountered.
14	- An attempt was made to execute a READ statement. However, the data item described by RELATIVE KEY is too small to accommodate the relative record number.
16	- An attempt was made to execute a READ statement although an AT END condition had already been encountered.

I-O status	Meaning
	Execution unsuccessful: invalid key condition
22	- Duplicate key An attempt was made to execute a WRITE statement which generated a duplicate key within the relative file.
23	- Record not located An attempt was made using a READ, START, DELETE or REWRITE statement with the aid of a key to access a data record not contained in the file.
24	- Boundary values exceeded An attempt was made to execute a WRITE statement outside the area set by the system, or a WRITE statement in sequential access mode with a relative record number so large that it does not fit in the data item defined with the RELATIVE KEY clause.
	Execution unsuccessful: unrecoverable error
30	- No further information regarding the I-O operation is available.
35	- An attempt was made to execute an OPEN statement for a nonexistent file.
38	- An attempt was made to execute an OPEN statement for a file previously locked by the LOCK phrase.
39	- The OPEN statement was unsuccessful because a a) One or more of the operands FCBTYPE, RECFORM or RECSIZE were specified in the FILE command with values deviating from the corresponding explicit or implicit program specifications (FCBTYPE must be PAM, RECFORM and RECSIZE should not be specified at all); or b) Record size error occurred for input data (catalog check); or c) The record size is greater than the BLKSIZE entry in the FILE command.

I-O status	Meaning
	Execution unsuccessful: logical error
41	- An attempt was made to execute an OPEN statement for a file which was already open.
42	- An attempt was made to execute a CLOSE statement for a file which was not open.
43	- For ACCESS MODE IS SEQUENTIAL: The most recent I-O statement executed prior to a DELETE or REWRITE statement was not a successfully executed READ statement.
46	- An attempt was made to execute a READ statement for a file in INPUT or I-O mode. However, there is no next valid record since: a) the preceding START statement terminated abnormally, or b) the preceding READ statement terminated abnormally without causing an AT END condition.
47	- An attempt was made to execute a READ or START statement for a file not in INPUT or I-O mode.
48	- An attempt was made to execute a WRITE statement for a file not in OUTPUT, I-O or EXTEND mode.
49	- An attempt was made to execute DELETE or REWRITE statement for a file not in I-O mode.
	Other unsuccessful executions
90	- System error; no further information available regarding the cause.
91	- System error; a system call terminated abnormally, e.g. password error.
93	- For simultaneous processing only (see "COB1 User's Guide" [7]): The I-O statement could not terminate normally because a different task is accessing the same file and the access operations are incompatible.
94	- For simultaneous processing only (see "COB1 User's Guide" [7]): Deviation from call sequence READ - REWRITE/DELETE.

4.2 LANGUAGE ELEMENTS OF THE ENVIRONMENT DIVISION

4.2.1 INPUT-OUTPUT SECTION

Function

The Input-Output Section covers the definition of each file, names its external devices, assigns each file to one or more input/output devices, and provides also the information needed for an efficient transmission of data between the input/output devices and the object program.

This Section is divided into two paragraphs:

the FILE-CONTROL paragraph, which names the files used in the program and assigns each file to external devices, and

the I-O-CONTROL paragraph, which indicates special input/output techniques.

Format

A margin indication

<u>INPUT-OUTPUT SECTION.</u>

[<u>FILE-CONTROL.</u> [file-control-entry.]...]

[<u>I-O-CONTROL.</u> [input-output-control-entry.]]

Programming considerations

1. All sections and paragraphs must begin at margin A.
2. The entire Input-Output Section is optional.

4.2.1.1 FILE-CONTROL Paragraph

Function

In the FILE-CONTROL paragraph each file is given a name and is then assigned to one or more I/O devices. This information specifies how the data is organized and how it is to be accessed.

Format

A B margin indication

FILE-CONTROL

SELECT clause
ASSIGN clause
RELATIVE KEY clause
[ORGANIZATION clause]

[ACCESS MODE clause]

[RESERVE clause]

[FILE STATUS clause]

Rules

1. The heading of the FILE-CONTROL paragraph must be written starting at margin A. All subsequent entries must be written starting an margin B.

2. The SELECT clause must be the first entry in the FILE-CONTROL paragraph. It is followed immediately by the ASSIGN clause. All other clauses may appear in any order.

In the pages that follow, the SELECT and ASSIGN clauses are described first, followed by the remaining clauses in alphabetical order.

SELECT CLAUSE

Function

The SELECT clause is used to give a name to each file in the program.

Format 1 names all files except sort-files.

Format 2 see chapter "Sort-Merge Feature".

Format 1

SELECT file-name

Rules

1. Each file-name used in a program may appear once only in the SELECT clause.
   File-name refers to the name with which a file is referenced in the source program (internal file-name).

2. Each file specified in a SELECT clause must have a file description (FD) entry in the Data Division.

3. Up to 254 files may be specified in the SELECT clauses within a source program.

4. The first 8 characters of file-name are used as a file link name, unless the COBRUN operand LINK was specified when the file was compiled (see ASSIGN clause and "COB1 User's Guide" [7]: COBRUN Operand). Since the link name cannot have more than 8 positions, file-name must be unique in these positions. If file-name is longer, it will be truncated for the link name. If it then ends with a hyphen in position 8, a warning is issued during compilation and the hyphen is converted into a number character (#).

5. If no FILE command is given for a file when the program is executed, file-name is used as a BS2000 file name in a FILE macro. In this case, all characters of file-name (up to 30) are used.

ASSIGN CLAUSE

Function

The ASSIGN clause assigns an external device to a file of the COBOL program. One ASSIGN clause is required for each file in the program.

Format 1 is used for all files except sort-files.

Format 2 see chapter "Sort-Merge Feature".

Format 1

```
ASSIGN TO [integer][implementor-name-1][implementor-name-2]...
```

Rules

1. The implementor-name designates a disk-storage unit and must be specified as follows:

   either

   a) class-[device-][organization-] name

   or

   b) [[class-] device- [organization-]] name-1

   (b) can only be specified in connection with COBRUN LINK; see "COB1 User's Guide" [7].

   class is a two-character field (optional in b only) that identifies a device class. Permitted: DA.

   device is an optional field that indicates a symbolic device name, i.e. the type of device to which the file is to be assigned. Permitted: DISC 1). The device assigned is a disk storage unit.

1) The DISC entry is used only for checking the validity of block/record length at compile time and does not affect the actual allocation of devices at object time. At object time, it is the user's responsibility to provide correct device assignments.

   organization is an optional single-character field specifying the file organization. The following character may be used:

   R for files with relative organization.

name is a six-character field that gives the symbolic name of the device to which the file is assigned. It must have the following format:

SYSnnn (where nnn is an integer between 000 and 244)

SYSnnn is not evaluated. The required LINK name is taken from the first 8 characters of the file-name given in the SELECT clause.

name-1 in (b) represents the required link name (see the description of COBRUN LINK in the "COB1 User's Guide" [7]).

2. **integer** indicates the number of devices assigned to the file. It is present only for reasons of compatibility, and is treated by the compiler as comments.

Programming considerations

1. Access can occur sequentially, randomly or dynamically.

2. File organization may also be specified in the ORGANIZATION clause (q.v.).

ACCESS MODE CLAUSE

Function

The ACCESS MODE clause determines the manner in which the records of a file are to be accessed.

Format

```
                ┌SEQUENTIAL┐
ACCESS MODE IS ─┤RANDOM    ├
                └DYNAMIC   ┘
```

Rules

1. SEQUENTIAL means that data records are read or written sequentially, i.e. the next logical data record of the file is made available when executing a READ statement, or the next logical data record is placed on that file when executing a WRITE statement.

2. RANDOM means that data records are read or written on the basis of a key, i.e. access is made using the RELATIVE KEY.

3. DYNAMIC means that direct and sequential access may be mixed.

4. If DYNAMIC is specified, a READ statement with or without the INVALID KEY phrase will indicate random access. A READ statement with the NEXT phrase indicates sequential access.

5. If the ACCESS MODE clause is not supplied, ACCESS IS SEQUENTIAL is assumed.

FILE STATUS CLAUSE

Function

The FILE STATUS clause specifies a data item whose contents are used for interrogating the status of input/output operations during processing. In addition, by specifying a further item, the DMS code is made available.

Format

FILE <u>STATUS</u> IS data-name-1 [, data-name-2]

Rules

1. data-name-1 and data-name-2 may only have one qualifier.

2. data-name-1 and data-name-2 must be defined in the LINKAGE SECTION or WORKING-STORAGE SECTION of the Data Division.
   data-name-1 must be a two-byte numeric (DISPLAY only) or alphanumeric item.
   data-name-2 must be a 6-character group item in the form

   ```
   0m          data-name-2.
   0m+1        data-name-x PIC 9(3) COMP.
   0m+1        data-name-y PIC X(4).
   ```

3. If the FILE STATUS clause is specified, the runtime system

   a) moves the I-O status to data-name-1, and

   b) if data-name-2 is specified, writes data-name-2 as follows, depending on the value of data-name-1 and the DMS code:

      b1) If data-name-1 has the value 0, data-name-2 will not be given a new value, i.e. its contents are considered undefined.

      b2) If data-name-1 has a non-zero value and the DMS code is 0, the left portion (first two bytes) of data-name-2 are filled with 0 while the right portion (remaining 4 bytes) remain undefined. If, however, the DMS code is non-zero, the left portion of data-name-2 is given the value 128 and the right portion is given the DMS code (for further information see the DMS manuals).

   This takes place during execution of the statements OPEN, CLOSE, DELETE, READ or REWRITE, which reference this file, and prior to execution of each corresponding USE PROCEDURE (see "I-O Status").

ORGANIZATION CLAUSE

Function

The ORGANIZATION clause defines the logical structure of a file.

Format

[ORGANIZATION IS] RELATIVE

Rules

Unlike Standard COBOL where the file organization must be specified in this clause, it is also possible to enter the file organization optionally in the ASSIGN clause, using the organization field of the implementor-name (see "ASSIGN Clause").

The entry can also be made in both places, in which case it must be identical.

If omitted altogether, ORGANIZATION IS SEQUENTIAL is assumed.

Programming consideration

File organization is defined at file construction time and is not, therefore, subject to modification.

RELATIVE KEY CLAUSE

The RELATIVE KEY clause specifies the key field whose contents are used for retrieving or placing a logical record in a relative file.

Format

```
RELATIVE KEY IS data-name
```

Rules

1. data-name may have one qualifier only.
2. data-name must not be subscripted or indexed.
3. The data item referenced by data-name must be defined as an unsigned integer.
4. data-name must not be supplied within the record description entry of the related file.
5. The data item referenced by data-name is used for passing on a relative record number from the user to the input/output system (FCP).

   All data records which exist on a file are uniquely identified by their relative record numbers. The relative record number of a given record specifies the logical position of that record within the file record sequence. The first logical record is relative record number one (1), and the following logical records have the relative record numbers 2, 3, 4... .

Programming consideration

When referencing a file by a START statement, the RELATIVE KEY clause must be specified for that file.

RESERVE-CLAUSE

Function

The RESERVE clause allows the user to modify the number of input/output areas (buffers) the compiled program will be allocated by the compiler

Format

RESERVE {integer } {AREA }

Rules

1. integer must be equal to 1.

2. RESERVE NO ALTERNATE AREA
   One input/output area is reserved.

4.2.1.2 I-O-CONTROL-Paragraph

Function

The I-O-CONTROL paragraph defines the events at whose occurrence restart points are to be established, and it specifies the memory area which is to be shared by the various files. Also, it indicates the location of files on multiple-file reels, and defines special input-output conditions.

Format

```
A    B      margin indication
|    |
↓    ↓
I-O-CONTROL.

    [RERUN clause]...

    [SAME AREA clause]... .
```

Rule

I-O-CONTROL must be written starting at margin A. All subsequent entries must be written in Area B.

For better readability, the clauses are described in the following subsections in alphabetic order..

RERUN-CLAUSE

Function

A RERUN clause indicates where and when restart-point records are to be issued. A restart-point record describes the status of an object program at a specified point during program execution.
It is produced automatically by the operating system upon the request of the object program and contains all information necessary to restart the program from that point. The RERUN clause controls such requests by the COBOL object program.

Format

```
RERUN [ON implementor-name]

           ┌integer-1 RECORDS OF data-name-2┐
     EVERY ┤integer-2 CLOCK-UNITS           │
           └condition-name                  ┘
```

Rules

1. implementor-name has the following format:

   SYSnnn.

   where: 000 ≤ nnn ≤ 244

   nnn determines the processing mode.

   If nnn ≤ 200, restart points are written alternately to two restart files. This makes it possible to restart from the two most recent restart points.

2. implementor-name designates the file to which the restart points are to be output.

3. Subject to the following restrictions, more than one RERUN clause may be specified for the same file-name-2.
   When specifying more than one integer-1 RECORDS phrase, the same file-name-2 must appear only once in them.

4. When implementor-name is specified, restart points are written as follows:

   a) If implementor-name is assigned to a file-name from the SELECT clause, the file involved must be a tape file. The effect on the generation of the restart point is the same as if the file-name had been specified in the RERUN clause.

   b) If implementor-name is not assigned to a file from a SELECT clause, a disk-storage file must be assigned at object time. Only in this case will restart point records be written to disk storage.

5. Restart points are requested by specifying integer-1 RECORDS. They are written to the file specified by file-name-1 or implementor-name, respectively, whenever integer-1 records have been processed. File-name-2 may be either an input or output file with any organization or type of access; it must not be referenced in the USING/GIVING phrase or in an INPUT/OUTPUT procedure on sort (see chapter "Sort-Merge Feature").

6. The CLOCK-UNIT phrase is treated by the compiler as comments.

7. The condition-name entry is likewise treated by the compiler as comments.

4

SAME AREA CLAUSE

Function

The SAME AREA clause indicates that two or more files are to share a specified input/output area during program execution.

Format 1 of the SAME AREA clause specifies all files except sort-files unless RECORD is supplied.

Format 2 see chapter "Sort-Merge Feature".

Format 1

SAME [RECORD] AREA FOR file-name-1 {file-name-2}...

Rules

1. More than one SAME AREA clause may be included in a program. In this case, the following must be observed:

   A specific file-name must not appear in more than one SAME AREA clause. The same is true of the SAME RECORD AREA clause.

   A specific file-name may concurrently appear in a SAME AREA clause and a SAME RECORD AREA clause. In this case, all file-names appearing in the SAME AREA clause must also appear in the SAME RECORD AREA clause. The SAME RECORD AREA clause may also contain other file-names that do not appear in the SAME AREA clause.

2. The SAME AREA clause indicates that the specified files (no sort-files) are to share the input/output areas assigned to them.

3. The SAME RECORD AREA clause indicates that the specified files are to share the same storage area for processing the current logical record.

   A logical record in the SAME RECORD AREA clause is considered a logical data record of all files opened for OUTPUT whose names are supplied in that SAME RECORD AREA clause. It is also a logical data record of the file (in this clause) from which the most recent input occurred. This is equivalent to an implicit overwriting of all data record areas, where the data records are aligned on the leftmost character position.

Programming considerations

1. If SAME AREA is used, only one file may be open at any give time.

2. If the RECORD phrase is used, all specified files may be open at the same time.

4.3 LANGUAGE ELEMENTS OF THE DATA DIVISION

4.3.1 FILE SECTION

Function

The FILE SECTION defines how the files are set up. Each file is defined by a file description entry and one or more record description entries. Record description entries are written immediately following the file description entry.

Format

```
A       margin indication
|
v
FILE SECTION.

[file description entry.    {record description entry}...]...
```

File description entries are discussed in the pages which follow. Record description entries are discussed in chapter 2 as well as in the pages which follow.

4.3.1.1 File Description (FD) Entry

Function

The file description (FD) entry specifies the physical setup, the identification, and the record names of a given file.

A file description entry must be written for each file to be processed by the program. The information contained in this entry generally pertains to the physical aspects of the file, that is, the description of the data as it appears on the input or output medium.

Format

```
A  B      margin indication
|  |
↓  ↓
FD file-name

  [BLOCK CONTAINS clause]

  [RECORD clause]

  [RECORDING MODE clause]
   LABEL RECORDS clause
  [DATA RECORDS clause].
```

Rules

1. The level indicator FD is located at the beginning of a file description entry, and must precede the file-name.

2. The file-name must be identical to the file-name given in a SELECT clause.

3. The clauses which follow the file-name are optional except for the LABEL RECORDS clause. The order in which they appear is immaterial.

4. The file description entry is terminated by a period.

5. If the LABEL RECORDS clause is omitted, LABEL RECORD IS STANDARD is assumed.

6. Table 4-2 provides a summary of the functions of the clauses used in file description entries.

   The formats and functions of these clauses are described on the following pages.

Programming consideration

The file description entry must be followed by one or more record description entries.

Clause	Function
BLOCK CONTAINS	Specifies physical block length
DATA RECORDS	Indicates the names of the data records in the file
LABEL RECORDS	Gives the names and values of the label records contained in the file.
RECORD	Specifies logical record size
RECORDING MODE clause	Specifies the format of the logical records

Table 4-2 Functions of file description clauses

4

BLOCKS CONTAINS CLAUSE

Function

The BLOCK CONTAINS clause specifies the maximum size of a physical block.

Format

BLOCK CONTAINS [integer-1 TO] integer-2 CHARACTERS

Rules

1. integer-1 and integer-2 must not be greater than 32763.
2. To calculate the block size, the larger of integer-1 and integer-2 is used (see "Structure of a Relative File" in "COB1 User's Guide" [7]).
3. integer-2 must not be less than the record length of the file.
4. If this clause is omitted, the compiler assumes that the record length of the file is integer-2.

DATA RECORDS CLAUSE

Function

The DATA RECORDS clause is used only for documentation. It specifies the names of the data records in a file.

Format

```
      ┌RECORD  IS ┐
DATA  ┤           ├ data-name-1 [data-name-2]...
      └RECORDS ARE┘
```

Rules

1. data-name-1, data-name-2 etc. are the names of data records. Each of these records must be preceded by the level number 01 in the file description entry.

2. The presence of more than one data-name indicates that the file contains more than one type of data record. These records may be of differing formats but must be of the same size. The order in which they are listed is immaterial.

LABEL RECORDS CLAUSE

Function

The LABEL RECORDS clause specifies whether labels are present; and if labels are present, this clause identifies them.

Format

$$\underline{\text{LABEL}} \left\{ \begin{array}{l} \underline{\text{RECORD}}\ \text{IS} \\ \underline{\text{RECORDS}}\ \text{ARE} \end{array} \right\} \underline{\text{STANDARD}}$$

Rule

The STANDARD phrase specifies that labels are present for the file and that these labels are in accordance with system conventions (see [2]).

RECORD CLAUSE

Function

The RECORD clause specifies the size of the data records in a file.

It indicates fixed-length records by specifying the number of character positions in a record.

The length of each data record is precisely defined by its record description entry. If the RECORD clause is specified, the record length is compared with the entry in this clause (see also RECORDING MODE Clause).

Format

<u>RECORD</u> CONTAINS integer-1 CHARACTERS

Rules

1. The number of character positions in each record description entry must be equal to integer-1.

2. integer-1 must be at least 1 and can be as large as 32767.

RECORDING MODE CLAUSE

Function

The RECORDING MODE clause specifies the format of the logical records in a file.

Format

RECORDING MODE IS F

Rule

F

indicates fixed-length records. It may be specified only when all the logical records in a file have the same length.

a) No record description for the file is permitted to contain an OCCURS clause with DEPENDING ON phrase.

b) Blocks may contain more than one record; in this case, a fixed number of records per block is implied.

c) If more than one record description is given for a file description (FD) entry, all records in the file must have the same length.

d) In F-mode blocks and records, no block-length fields (BLF) and record length fields (SLF) are used.

4

4.4 LANGUAGE ELEMENTS OF THE PROCEDURE DIVISION

4.4.1 INVALID KEY Condition

The INVALID KEY condition may occur after the execution of a START, READ, WRITE, REWRITE or DELETE statement. Details about the occurrence of this condition are described under each of these statements.

If an INVALID KEY condition was encountered, the File Control Processor performs the following actions in the specified order.

1. A value is entered into the FILE STATUS data item, if specified for this file, to indicate the INVALID KEY condition (see "FILE STATUS Clause").

2. If the statement causing an invalid key condition includes an INVALID KEY phrase, then the imperative statement following INVALID KEY is executed. Any USE procedure supplied for that file will not be executed.

3. If no INVALID KEY phrase is present, but a user procedure exists for that file, the USE procedure is executed. This may be an explicitly or implicitly declared user procedure.

4. If neither the INVALID KEY phrase nor a USE procedure is present for this file, the invalid key condition causes the program to terminate abnormally.

If an invalid key condition occurs, the execution of the input/output statement which recognized the condition is unsuccessful, and the file is in the same condition as it was before this input/output statement was executed.

4.4.2 Input-Output Statements

In COBOL, input and output is record-oriented. Thus the READ, WRITE, DELETE and REWRITE statements process records. The COBOL user is therefore only concerned with the processing of single records. The following operations are performed automatically: moving data into input/output areas (buffers) and/or internal storage, validity checking, and error correction (where feasible).

Summary

Statement	Function
CLOSE	Terminates processing of a file.
DELETE	Removes a data record.
OPEN	Opens a file for processing.
READ	Makes a record available from an input or input/output file.
REWRITE	Replaces a data record on a disk-storage file by a specified data record.
START	Positions within a file.
USE	In addition to input/output statements, the USE statement may be supplied to indicate label and error handling procedures (see "Declaratives Subdivision of the Procedure Division").
WRITE	Releases a record to an input/output or output file.

CLOSE STATEMENT

Function

The CLOSE statement terminates the processing of files, with optional rewind and/or lock where applicable.

Format

```
CLOSE file-name-1 [WITH LOCK]

     [file-name-2 [WITH LOCK]]...
```

Rules

1. file-name-1, file-name-2,... specify files on which the CLOSE statement is to operate.

2. file-name-1, file-name-2,... must be described in the file description (FD) entry in the Data Division of the program.

3. The files referenced in the CLOSE statement need not all have the same organization or access.

4. A CLOSE statement may be executed only on an open file.

5. The following paragraphs describe the meanings of the various CLOSE statements in random access file processing.

   The results of executing each CLOSE phrase are summarized in Table 4-3. The definitions of the symbols used in it are given below.

   K - **Standard Close File**
   The standard system close procedures are implemented.

   L - **Standard File Lock**
   The compiler ensures that the file cannot be reopened while the program is running.

CLOSE phrase	File type	
	Single-volume	Multi-volume
CLOSE	K	K
CLOSE WITH LOCK	K, L	K, L

   Table 4-3 Relationship of types of random-access files and CLOSE statement phrases

6. After execution of a CLOSE statement, the contents of the data-name specified in the FILE STATUS clause will be updated (see also "FILE STATUS Clause").

Programming considerations

1. After successful execution of a CLOSE statement, the record area assigned to the file is no longer accessible, whereas availability of the record area after unsuccessful execution of a CLOSE statement is unpredictable.

2. All files remaining open when a task has finished are closed.

3. If a CLOSE statement was executed for a file, the next input/output statement to be executed must be an OPEN statement.

4. The LOCK phrase signifies that the file cannot be reopened within the same program run.

DELETE STATEMENT

Function

A DELETE statement logically removes a data record from a disk-storage file.

Format

DELETE file-name RECORD [INVALID KEY imperative-statement]

Rules

1. file-name must be the name of a disk-storage file.

2. For a file whose access mode is sequential, the INVALID KEY phrase of the DELETE statement must not be supplied.

3. For a file whose access mode is other than sequential, the INVALID KEY phrase is mandatory if no appropriate USE procedure is supplied.

4. For a file whose access mode is sequential, the last input/output statement for the associated file, which preceded the DELETE statement must have been a successfully completed READ statement. The RELATIVE KEY may not be changed between reading and deletion. Execution of the DELETE statement causes the data record read by the preceding READ statement to be deleted from that file.

5. For a file with random or dynamic access, the File Control Processor deletes the data record identified by the contents of the data item specified in the RELATIVE KEY clause of the file. If the data record referenced by the key does not exist on the file, an INVALID KEY condition occurs (see "INVALID KEY Condition").

6. After execution of the DELETE statement, the contents of the data item specified in the FILE STATUS clause for the file is updated (see "FILE STATUS Clause").

Programming considerations

1. The file referenced in the DELETE statement must be in I-O open mode during the execution of this statement (see also "OPEN Statement").

2. After successful execution of the DELETE statement, the identified data record is deleted from the file.

3. Execution of a DELETE statement does not affect the contents of the data record area associated with the file.

OPEN STATEMENT

Function

The OPEN statement opens files for processing.

Format

```
      ⎧INPUT file-name-1 [file-name-2]...  ⎫
OPEN  ⎨OUTPUT file-name-3 [file-name-4]... ⎬...
      ⎩I-O file-name-5 [file-name-6]...    ⎭
```

Rules

1. file-name-1, file-name-2, file-name-3,... must be defined by a file description (FD) entry in the Data Division.

2. At least one of the INPUT, OUTPUT or I-O options must be used. If two or more such options are used, they may appear in any order. Two or more file-names may be supplied for one option; however, the same file-name must not appear more than once in a given OPEN statement.

3. The files specified in a given OPEN statement may have different organizations or access modes.

4. After successful execution of an OPEN statement, the file specified by file-name is available in the open mode.

5. After successful execution of an OPEN statement, the associated record area is available to the program.

6. Before successful execution of an OPEN statement for a file, no statement may be executed that would either explicitly or implicitly reference that file.

7. INPUT means that the file is to be processed as an input file (input mode).

8. OUTPUT indicates that the file is to be processed as an output file (output mode).

9. I-O indicates that input and output operations (i.e. reading, writing, and updating operations) are to be performed on the file.

   Since this phrase implies the existence of the file, it cannot be used if the file is being initially created (update mode).

10. After execution of an OPEN statement, the contents of the data-name specified in the FILE STATUS clause (if specified) will be updated (see also "FILE STATUS Clause").

Programming considerations

1. All of the INPUT, OUTPUT, or I-O phrases may be used, within the same program, in different OPEN statements for a given file. Before executing another OPEN statement after the logically first one for the same file in a program, a CLOSE statement must have been performed; at this time, LOCK may not appear in the CLOSE statement.

2. Table 4-4 lists the statements permissible in OPEN mode (X = "permitted").

ACCESS clause	Statement	OPEN mode		
		INPUT	OUTPUT	I-O
SEQUENTIAL	READ	X		X
	WRITE		X	
	REWRITE			X
	START	X		X
	DELETE			X
RANDOM	READ	X		X
	WRITE		X	X
	REWRITE			X
	START			
	DELETE			X
DYNAMIC	READ	X		X
	WRITE		X	X
	REWRITE			X
	START	X		X
	DELETE			X

Table 4-4 Valid input-output statements in each OPEN mode

READ STATEMENT

Function

The READ statement places

- the next data record in the file when access is sequential
- a particular data record in a disk file when access is random

at the disposal of the program.

Format 1 is used for the sequential retrieving of records from any file with sequential or dynamic access mode.

Format 2 is used for the random (RELATIVE KEY) retrieving of records from any file with random or dynamic access mode.

The addendum WITH NO LOCK in both formats is effective when simultaneous processing is used; it is described in the "COB1 User's Guide" [7] (see under "SHARUPD").

Format 1

```
READ file-name [WITH NO LOCK] [NEXT] RECORD [INTO identifier]

                              [AT END imperative-statement]
```

Rules for format 1

1. file-name must appear in a file description entry in the Data Division of the program.

2. NEXT must be specified when data records are to be read from a file sequentially using dynamic access mode.

3. AT END must be specified when there is no USE procedure for the file.

4. The READ statement makes available to the program a record from the file specified by file-name, unless an exceptional condition exists. Here, the data record is read into the input area which was specified by the record description entry in the file description. It remains available there until the next input/output statement is executed for that file.

5. A READ statement with INTO phrase specified causes the following to happen:

   a) The data record is read into the input area (as described under 4 above).

b) This data record is then moved from the input area to the area specified by identifier (implicit MOVE). The MOVE operation takes place according to the rules for MOVE statements without CORRESPONDING phrase. After the READ statement with INTO phrase has been successfully executed, the data record is available both in the input area and in the area specified by the identifier.

   If the operation described in a) was unsuccessful, no MOVE takes place.

   The index for identifier is calculated after the operation described in a) is executed but immediately prior to the implicit MOVE.

6. When executed, a READ statement causes the data item specified in the FILE-STATUS clause for the relevant file to be updated (I-O Status; see also "FILE STATUS Clause").

7. Following an unsuccessful attempt to perform a READ statement, the contents of the record area associated with the file as well as the position within the file are unpredictable.

Programming considerations for format 1

1. An OPEN statement with the INPUT or I-O phrase must be executed for a file before the READ statement can be executed.

2. When a file contains more than one logical record type, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. Only the information in the current record is accessible. Thus, no statement in the Procedure Division can refer to a different logical record.

3. If a READ statement is executed, and there is no further data record on the file, then an end-of-file condition occurs, and the execution of that READ statement is considered unsuccessful (see also "FILE STATUS Clause").

4. If the end-of-file condition is encountered, the following steps are performed, in the order given below:

   a) The contents of the data item from the FILE STATUS clause, if specified for the file, is supplied with a value indicating the AT END condition (see "FILE STATUS Clause").

   b) If AT END is specified in the READ statement, control is transferred to the imperative statement following the AT END phrase. Any USE statement declared for that file will not be executed.

   c) If the AT END phrase is omitted from the READ statement, either an explicit or implicit USE procedure must be declared for the file. When the AT END condition occurs, that USE procedure will then be executed.

   When the AT END condition occurs, execution of the input/output statement must be considered to be unsuccessful.

5. If the RELATIVE KEY phrase is present for a file, successful execution of a format-1 READ statement is followed by updating the contents of the RELATIVE KEY data item so that it contains the relative record number of the data record made available.

6. After occurrence of the AT END condition, no format-1 READ statement must be issued for the file before one of the following steps is taken.

   a) A successful CLOSE statement, followed by a successful OPEN statement for that file.

   b) A successful START statement for that file.

   c) A successful format-2 READ statement for that file.

Format 2

```
READ file-name [WITH NO LOCK] RECORD [INTO identifier]
                              [INVALID KEY imperative-statement]
```

Rules for format 2

1. file-name must be defined by a file description entry in the Data Division of the program.

2. If no USE procedure is declared for the file, AT END must be supplied in the READ statement.

3. The READ statement makes available to the program a record from the file specified by file-name, unless an exceptional condition exists. Here, the data record is read into the input area which was specified by the record description entry in the file description. It remains available there until the next input/output statement is executed for that file.

4. A READ statement with INTO phrase specified causes the following to take place (in order given below):

   a) The same READ statement is executed without INTO phrase.

   b) The current data record is moved from the input area to the area specified by identifier (implicit MOVE). The MOVE operation takes place according to the rules for MOVE statements without CORRESPONDING phrase. After the READ statement with INTO phrase has been successfully executed, the data record is available both in the input area and in the area specified by identifier.

      If the READ statement described in a) was unsuccessful, no move operation takes place.

      The index for identifier is calculated after the READ statement described in a) is executed but immediately prior to the implicit MOVE.

5. Execution of a READ statement causes the contents of the data item specified in the FILE STATUS clause of the related file description entry to be updated (see also "FILE STATUS Clause").

6. After an unsuccessful attempt to perform a READ statement, the contents of the record area associated with the file as well as the position within the file are unpredictable.

7. If a file is in random or dynamic access mode, the contents of the data item associated with the RELATIVE KEY must be set to the required value before the READ statement is executed (see "RELATIVE KEY Clause").

8. When a file is being read randomly, a search is made for a data record whose relative record number is equal to the value of the RELATIVE KEY item of that file. If no such data record exists on the file, an INVALID KEY condition occurs (see also "INVALID KEY Condition").

Programming considerations for format 2

1. An OPEN statement with the INPUT or I-O phrase must be executed for a file before the READ statement can be executed.

2. When a file contains more than one logical record type, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. Only the information in the current record is accessible. Thus, no statement in the Procedure Division can refer to a different logical record.

REWRITE STATEMENT

Function

The REWRITE statement replaces a logical record on a disk-storage file.

4

Format

```
REWRITE record-name [FROM identifier]
        [INVALID KEY imperative-statement]
```

Rules

1. record-name and identifier must not refer to the same storage area.

2. record-name must be defined in a file description (FD) entry in the Data Division of the program.

3. The INVALID KEY phrase is not permitted for a REWRITE statement referring to a file whose access mode is sequential.

4. The INVALID KEY phrase must be specified for a REWRITE statement, unless an applicable USE procedure was declared.

5. record-name identifies the record to be replaced.

6. Execution of a REWRITE statement with the FROM phrase specified corresponds to the following statements:

   MOVE identifier TO record-name
   REWRITE record-name.

   The contents of the storage area described by record-name do not effect execution of the REWRITE statement until the implicit MOVE statement has been performed.

   Therefore, when using the FROM phrase, data is transferred from identifier to record-name and is then released to the appropriate file. Identifier may be used for referencing any data outside the current file description entry.

   Transfer of data through the implicit MOVE statement takes place in accordance with the rules for a MOVE statement without the CORRESPONDING phrase. After the REWRITE statement is executed, the contents of the record is still available in the area specified by the "identifier" although it is no longer available in the area specified by record-name.

7. The following rules apply to all files whose access mode is sequential:

   a) A REWRITE statement must be preceded by a successful READ statement as the last input/output statement for the associated file.

   b) Execution of the REWRITE statement causes the record accessed by the preceding READ statement to be replaced on that file.

8. For any file with an access mode other than sequential, the RELATIVE KEY must be provided with a corresponding value before the REWRITE statement is executed.

9. For files whose access mode is random or dynamic, the data record referenced by the contents of the RELATIVE KEY data item associated with the file will be replaced.

10. If the file does not contain a data item corresponding to the contents of the RELATIVE KEY item, the INVALID KEY condition occurs (see "INVALID KEY Condition").

11. The data record rewritten by a successful REWRITE statement is no longer accessible in the record area; an exception to this rule is the use of the SAME RECORD AREA clause. In this case, the data record is available to all other files specified in the SAME RECORD AREA clause as well as to the current file.

12. Execution of the REWRITE statement causes the contents of the data item that was specified in the FILE STATUS clause of the related file description entry to be updated (see also "FILE STATUS Clause").

Programming considerations

1. The file associated with record-name must be a disk-storage file and must have been opened I-O at the time the REWRITE statement is executed.

2. The data record addressed by the data record name must be of the same length as the data record to be replaced on the file.

3. When a data record is being rewritten, the first byte of the record must not contain the value X'FF' as this would effect an illogical deletion of the record.

START STATEMENT

Function

The START statement defines the logical starting point, within a file, for subsequent sequential read operations.

Format

```
                                      ┌     ┌IS EQUAL TO                ┐             ┐
                                      │     │IS =                       │             │
                                      │     │IS GREATER THAN            │             │
                                      │     │IS >                       │             │
                                      │     │IS NOT LESS THAN           │             │
START file-name [WITH NO LOCK]        │ KEY ┤IS NOT <                   ├ data-name   │
                                      │     │IS LESS THAN               │             │
                                      │     │IS <                       │             │
                                      │     │IS LESS THAN OR EQUAL TO   │             │
                                      │     │IS NOT GREATER THAN        │             │
                                      └     └IS NOT >                   ┘             ┘
      [INVALID KEY imperative-statement]
```

The addendum WITH NO LOCK is effective when simultaneous processing is used; it is described in the "COB1 User's Guide" [7] (see under "SHARUPD").

Rules

1. Relational operators, when specified, are required separators.

2. file-name must refer to a file with sequential or dynamic access mode.

3. data-name may be qualified.
   It must be the name of the RELATIVE KEY data item declared for that file.

4. The INVALID KEY phrase must be present if no applicable USE procedure has been declared for that file.

5. The type of comparison is specified by the relational operator in the KEY phrase. The position of each record in the file specified by file-name is compared with the contents of the data item referenced by data-name (RELATIVE KEY):

   a) With the relational operators EQUAL, GREATER, GREATER OR EQUAL, NOT LESS and their equivalents, positioning takes place in the file to the first record that satisfies the relational operator.
   b) With the relational operators LESS, LESS OR EQUAL, NOT GREATER and their equivalents, positioning takes place in the file to the last record that satisfies the relational operator.
   c) If no record in the file satisfies the relation condition, an INVALID-KEY condition occurs and the START statement is terminated abnormally (see "INVALID-KEY condition").

6. If the KEY phrase is omitted from the START statement, the comparison described under 5) will use the data item specified by RECORD KEY for that file, and the relational operator "=" will be assumed.

7. After Execution of a START statement, the contents of the FILE STATUS data item of that file will be updated (see also "FILE STATUS Clause").

8. The file specified by file-name must have been opened at the time the START statement is executed (see "OPEN Statement").

USE STATEMENT

Function

The USE statement introduces declarative procedures and defines the condition for their execution. The USE statement itself, however, is not executed.

Format 1 see "Sequential File Organization".

Format 2 introduces procedures to be executed if an input/output error occurs in a file.

Format 2

```
                      ┌ERROR    ┐
USE AFTER STANDARD   {          } PROCEDURE
                      └EXCEPTION┘

         ┌file-name-1 [file-name-2]...┐
         │INPUT                       │
    ON  {                              }
         │OUTPUT                      │
         └I-O                         ┘
```

Rules

1. The ERROR and EXCEPTION phrases are equivalent and may be used optionally.

2. file-name must be defined in a file description (FD) entry of the program's Data Division.

3. When the file-name phrase is specified, the error handling procedures are executed only for the files named. No further USE procedures are run for these files.

4. The specified procedures are executed by the system when an input/output error occurs during the execution of an input/output statement for a file.

5. Before execution of the user error routine, the standard system error routines for input/output error handling are executed.

6. After a USE procedure is executed, control passes to the calling routine.

7. INPUT indicates that the specified procedures are executed only for files opened in input mode (OPEN statement with the INPUT option specified).

8. OUTPUT indicates that the specified procedures are executed only for files opened in output mode (OPEN statement with the OUTPUT phrase specified).

9. I-O indicates that the specified procedures are executed only for files opened in update mode (OPEN statement with the I-O option specified).

Programming considerations

1. The USE procedures are run when:

   a) an INVALID KEY or AT END condition occurs, provided that the input/output statement where this occurrence takes place does not include an INVALID KEY or AT END phrase.

   b) a severe error occurs (FILE STATUS CODE ≥ 30).

   If no corresponding USE procedure exists for the file when a) or b) occurs, the program will abort.

2. Files referenced either implicitly (INPUT, OUTPUT and I-O) or explicitly (file-name-1, file-name-2,...) in the USE statement need not have the same organization and access mode.

3. Within a USE procedure, there must be no reference to non-declarative procedures excepting the PERFORM statement.

   Reference to procedure-names which are subordinate to a USE statement may be made from another procedure, or from a non-declarative procedure, only by using a PERFORM statement.

4. File-name may not be used in more than one USE statement.

Example 4-1 (format 2)

Environment Division statements:

```
      SELECT MASTER-FILE ASSIGN DA-DISC-R-SYS001
      RELATIVE KEY RELKEY
      RESERVE 1 AREA
      FILE STATUS INDICATOR.
  .
  .
  .
```

Data Division entries:

```
  FILE SECTION.
  FD  MASTER-FILE LABEL RECORD STANDARD.
  01  RECORD-FORMAT
      03 CONTENTS                 PIC X(100).
  WORKING-STORAGE SECTION.
  77  INDICATOR                   PIC 99.
  77  RELKEY                      PIC 9(4).
  77  CLOSE-INDICATOR             PIC X VALUE "0".
  .
  .
  .
```

Procedure Division statements:

```
  PROCEDURE DIVISION.
  DECLARATIVES.
  INPUT-ERROR SECTION.
      USE AFTER ERROR PROCEDURE ON MASTER-FILE.
  STATUS-QUERY.
      IF INDICATOR NOT < 30
           GO TO UNRECOV-ERROR.
  END-OF-FILE.
      IF  INDICATOR = 10
           DISPLAY "END OF MASTER-FILE ENCOUNTERED."
           GO TO SPOOL-BACK.
  I-O-DUPKEY.
      IF  INDICATOR = 22
           DISPLAY "RECORD WITH KEY" RELKEY
           "ALREADY EXISTS"
           GO TO SPOOL-BACK.

  NON-EXISTENT.
      IF  INDICATOR = 23
           DISPLAY "RECORD WITH KEY" RELKEY
           "NON-EXISTENT"
           GO TO SPOOL-BACK.
```

```
UNRECOV-ERROR.
    DISPLAY "UNRECOVERABLE ERROR ("INDICATOR")"
        "FOR MASTER FILE".
    IF CLOSE-INDICATOR = "1"
        CLOSE MASTER FILE.
    DISPLAY "PROGRAM TERMINATED ABNORMALLY"
    STOP RUN.
SPOOL-BACK.
    EXIT.
END DECLARATIVES.
START.
    OPEN MASTER FILE.
    MOVE "1" TO CLOSE-INDICATOR.
    .
    .
    .
    MOVE "0" TO CLOSE-INDICATOR.
    CLOSE MASTER-FILE
```

WRITE STATEMENT

Function

The WRITE statement causes a data record to be released to an input/output or an output file.

Format 1 see "Sequential File Organization".

Format 2 of the WRITE statement must be used for disk-storage files with any organization other than sequential.

Format 2

```
WRITE record-name   [FROM identifier-1]
    [INVALID KEY   imperative-statement]
```

Rules

1. record-name and identifier-1 must not reference the same storage area.

2. record-name is the name of a data record in the FILE SECTION and must not be qualified by a file-name.

3. The INVALID KEY phrase must be specified for a file unless a USE procedure was declared.

4. The data record supplied by a WRITE statement is no longer available in the record area, unless the file related to the data record was specified in a SAME RECORD AREA clause or the execution of the WRITE statement was abnormally terminated as unsuccessful because of the occurrence of an INVALID KEY condition. The record is also available to those files which were referenced in a SAME RECORD AREA clause together with the specified file.

5. Execution of a WRITE statement with the FROM phrase is equivalent to the following statements:

   MOVE identifier-1 TO record-name.
   WRITE record-name.

   Data is transferred according to the rules for a MOVE statement without the CORRESPONDING phrase.

   The contents of the record area before execution of the implicit MOVE statement has no effect on the execution of the WRITE statement.

   After the WRITE statement is successfully executed, the information is still available in the area referred to by the identifier; however, as pointed out under Rule 4, this is not necessarily applicable to the data record area.

6. After execution of a WRITE statement, the value of the data item of any FILE STATUS clause existing for that file is updated (see "FILE STATUS Clause").

7. The WRITE statement is used to add data records to a disk-storage file.

8. If a file is opened in output mode (OPEN statement with OUTPUT phrase), the following should be noted:

   a) In sequential access mode, the WRITE statement causes output of a data record for creation of a new file. The first data record is assigned a relative record number of 1 (one) while the subsequent records are numbered 2, 3, 4,... If the RELATIVE KEY clause was.. specified, the File Control Processor will enter, during the execution of the WRITE statement, the relative record number in the RELATIVE KEY data item.

   b) In random or dynamic access modes (which are equivalent for OUTPUT), the value of the data item in the (here mandatory) RELATIVE KEY clause must be set by the user to equal the relative record number which is to be assigned to the data record contained in the record area. That data record is then output as the nth record of the file, where "n" is the value of the relative record number.

9. If a file is opened in update mode (OPEN statement with I-O option), and its access mode is either random or dynamic (which means the same in this case), the WRITE statement inserts data records in the associated existing file. The value of the data item of the (here mandatory) RELATIVE KEY clause must be set by the user to equal the relative record number which is to be assigned to the data record contained in the record area. As the WRITE statement is executed, the contents of the record with the appropriate record number is transferred to the File Control Processor.

10. The INVALID KEY condition is triggered by the causes listed in table 4-8.

11. Occurrence of an INVALID KEY condition indicates that the WRITE statement was unsuccessful; the contents of the data record area are still available, and any existing FILE STATUS data item in that file is set to a value indicating the cause of the INVALID KEY condition. The program resumes execution in accordance with the rules of the INVALID KEY condition.

Programming considerations

1. The file whose data record is referred to by the WRITE statement must have been opened as OUTPUT or I-O.

2. When executing a WRITE statement, the position of the data record to be output within the file is located by means of the contents of the RELATIVE KEY data item when the file is in random or dynamic access mode.

3. Before the WRITE statement is executed, the contents of the associated key field must be set accordingly (see "RELATIVE KEY Clause").

   A summary is given in Table 4-5.

4. A data record must not contain HIGH-VALUE in its first character, since HIGH-VALUE is used as an indicator for a dummy record.

Organization	Access	Open mode and action taken	Cause of invalid key condition
Relative	SEQUENTIAL	OUTPUT A record is added to a file to be newly created.	No space is available on the file for writing the record
	RANDOM or DYNAMIC	OUTPUT or I-O A record is added to an existing file.	The contents of the RELATIVE KEY data item specifies a data record already existing on the file, or no space is available on the file for writing the record.

Table 4-5 WRITE statement. Causes of INVALID KEY conditions

5 INDEXED FILE ORGANIZATION

5.1 FILE CONCEPTS

5

A file is a collection of data records that can be transferred to, or read from, a volume. The user defines the organization of the file as well as the mode and order of file processing.

The organization of a file describes its logical structure. There are sequential, indexed, relative, and direct types of file organization. The file organization which is defined at the time the file is constructed cannot be changed later on. This same rule applies to the record size of a particular file, as defined in the file and record description entries.

5.1.1 Indexed Organization

When indexed file organization is used, the position of each logical data record in the file is determined by indices, which are generated with the file and are maintained by the system.
These indices are based on keys to be supplied by the user. Indexed files must be assigned to disk-storage devices.

5.1.2 Sequential Access to Data Records of an Indexed File

Data records in a relative file may be sequentially created, read, updated, and added.

An indexed file can only be **created sequentially.** At the time an indexed file is created, the RECORD KEY clause must be specified. It is used for indicating the position of the key within the record. The records will appear in the file in the order in which they were written.

Reading or updating an indexed file sequentially requires that the RECORD KEY be supplied. The records are read in the order in which they were previously placed on the file. Consequently, this corresponds to the order of keys, which must be sorted in ascending sequence at file creation time.

5.1.3 Random Access to Data Records of an Indexed File

Data records of indexed files may be randomly read, updated, and added.

Reading or updating an indexed file randomly requires that the RECORD KEY clause be supplied. A record is considered "found" when the values of the key field and RECORD KEY coincide for that record. When randomly adding or updating a data record in an indexed file, the values of RECORD KEY and the key field of the record must be identical.

Reading of the data records is performed on the basis of a RECORD KEY. Records may be updated by reading a data record into a storage area, modifying the record in that area, and rewriting the record from that same area.

5.1.4 Dynamic/Extended Access to Data Records of an Indexed File

Alternation between sequential to random access.

5.1.5 Valid Access Methods

Types of file organization	Allowable access methods (X)			
and possible actions	Sequential	Random	Extended	Dynamic
Indexed files				
Record creation	X		X	X
Record retrieval	X	X	X	X
Record update	X 1)	X	X	X
Record addition	X	X	X	X

Table 5-1 Valid access methods

1) In sequential access, updating a data record means updating in place; i.e. a READ immediately followed by a REWRITE; this is allowable for disk-storage files only.

5.1.6 I-O Status

The I-O status is a value which is transferred to a two-character data item for the purpose of indicating to the COBOL program the status of an input/output operation. This takes place during execution of a CLOSE, DELETE, OPEN, READ, REWRITE, START or WRITE statement and prior to execution of any associated imperative statement and any corresponding USE AFTER STANDARD EXCEPTION procedure. The I-O status value is transferred only if the FILE STATUS clause is specified in the FILE-CONTROL paragraph (see FILE STATUS clause).

At the termination of an I-O operation the values of the I-O status have the following meaning:

1. Execution successful
   The I-O statement was processed normally.

2. Execution unsuccessful: AT END condition
   A READ statement terminated abnormally due to an AT END condition.

3. Execution unsuccessful: invalid key condition
   The I-O statement terminated abnormally due to an invalid key condition.

4. Execution unsuccessful: unrecoverable error
   The I-O statement terminated abnormally due to the occurrence of an unrecoverable error precluding further processing of the file. An unrecoverable error can not be rectified within the COBOL program.

5. Execution unsuccessful: logical error
   An I-O statement terminated abnormally due to an illegal sequence of I-O operations for this file.

6. Conflict between the permanent characteristics of a file and those described in the program
   The I-O statement terminated abnormally because the permanent file characteristics are incompatible with those given in the COBOL program.

7. System error
   The I-O statement terminated abnormally due to a system error.

The table below shows the values of the I-O status and their meanings:

I-O status	Meaning
	Execution successful
00	- The I-O statement terminated normally. No further information regarding the I-O operation is available.
04	- Record length conflict: A READ statement terminated normally. However, the length of the record read lies outside the limits defined in the record description entry for this file.
	Execution unsuccessful: AT END condition
10	- An attempt was made to execute a READ statement. However, the next logical record was nonexistent as the end-of-file was encountered.
16	- An attempt was made to execute a READ statement although an AT END condition had already been encountered.
	Execution unsuccessful: invalid key condition
21	- File sequence error in conjunction with ACCESS MODE IS SEQUENTIAL: a) The record key value was changed after the successful execution of a READ statement and prior to execution of the next REWRITE statement, or b) the ascending sequence of record keys was abandoned in consecutive WRITE statements.
22	- Duplicate key An attempt was made to execute a WRITE statement which generated a duplicate key within the relative file.
23	- Record not located An attempt was made using a READ, START, DELETE or REWRITE statement with the aid of a key to access a data record not contained in the file.
24	- Boundary values exceeded An attempt was made to execute a WRITE statement outside the area set by the system for an indexed file.

I-O status	Meaning
	Execution unsuccessful: unrecoverable error
30	- No further information regarding the I-O operation is available.
35	- An attempt was made to execute an OPEN statement for a nonexistent file.
38	- An attempt was made to execute an OPEN statement for a file previously locked by the LOCK phrase.
39	- The OPEN statement was unsuccessful because a) One or more of the operands FCBTYPE, RECSIZE, RECFORM, KEYPOS or KEYLENGTH were given values that conflict with the corresponding explicit or implicit program specifications; or b) Record size errors occurred with input files (catalog check); or c) The record size is greater than the BLKSIZE entry in the FILE command d) A RECSIZE ≠ 0 was specified in the FILE command for an output file with RECFORM=V.
	Execution unsuccessful: logical error
41	- An attempt was made to execute an OPEN statement for a file which was already open.
42	- An attempt was made to execute a CLOSE statement for a file which was not open.
43	- For ACCESS MODE IS SEQUENTIAL: The most recent I-O statement executed prior to a DELETE or REWRITE statement was not a successfully executed READ statement.
44	- Exceeding the area limits: An attempt was made to execute a WRITE or REWRITE statement. However, the length of the record exceeds the area allowed for this file.

I-O status	Meaning
46	- An attempt was made to execute a READ statement for a file in INPUT or I-O mode. However, there is no next valid record since the preceding READ statement terminated abnormally without causing an AT END condition since: a) The preceding START Statement terminated abnormally, or b) The preceding READ statement terminated abnormally but did not cause an AT END condition.
47	- An attempt was made to execute a READ statement for a file not in INPUT or I-O mode.
48	- An attempt was made to execute a WRITE statement for a file not in OUTPUT, I-O or EXTEND mode.
49	- An attempt was made to execute a REWRITE statement for a file no in I-O mode.
	Other unsuccessful executions
90	- System error; no further information available regarding the cause.
91	- System error; a system call terminated abnormally, e.g. password error.
93	- For simultaneous processing only (see "COB1 User's Guide" [7], "Simultaneous Processing"): The I-O statement could not terminate normally because a different task is accessing the same file and the access operations are incompatible.
94	- 1. For simultaneously processing only (see "COB1 User's Guide" [7], "Simultaneous Processing"): Deviation from call sequence READ - REWRITE/DELETE. 2. Record size is greater than the block size.

5.2 LANGUAGE ELEMENTS OF THE ENVIRONMENT DIVISION

5.2.1 INPUT-OUTPUT SECTION

Function

The INPUT-OUTPUT SECTION is used to define each file, to name its external devices, to assign each file to one or more input/output devices, and also to provide the information needed for an efficient transmission of data between the input/output devices and the object program.

This SECTION is divided into two paragraphs:

- The FILE-CONTROL paragraph, which names the files used in the program and assigns each file to external devices, and
- the I-O-CONTROL paragraph, which indicates special input/output techniques.

Format

A margin indication

INPUT-OUTPUT SECTION.

[FILE-CONTROL. [file-control-entry.]...]

[I-O-CONTROL. [input-output-control-entry.]]

Programming considerations

1. All sections and paragraphs must begin at margin A.
2. The entire INPUT/OUTPUT SECTION is optional.

5.2.1.1 FILE-CONTROL Paragraph

Function

The FILE-CONTROL paragraph is used to give each file a name, to assign it to one or more I/O devices and also to provide the information required for processing. This information indicates how the data is organized and how it is to be accessed.

Format

```
A    B      margin indication

FILE-CONTROL

      SELECT clause
      ASSIGN clause
      RECORD KEY clause
     [ORGANIZATION clause]

     [ACCESS MODE clause]

     [RESERVE clause]

     [FILE STATUS clause]
```

Rules

1. The heading of the FILE-CONTROL paragraph must be written starting at margin A. All subsequent entries must be written starting at margin B.

2. The SELECT clause must be the first entry in the file description entry. All clauses which follow the SELECT clause, with the exception of the ASSIGN clause, may appear in any order.

In the pages that follow, the SELECT and ASSIGN clauses are described first, followed by the remaining clauses in alphabetical order.

SELECT CLAUSE

Function

The SELECT clause is used to give a name to each file in the program.

Format 1 names all files except sort-files.

Format 2 see chapter "Sort-Merge Feature".

Format 1

SELECT file-name

Rules

1. Each file-name used in a program is only permitted to occur once in the SELECT clause. File-name refers to the name with which a file is referenced in the source program (internal file name).

2. Each file specified in a SELECT clause must have a file description (FD) entry in the Data Division of the source program.

3. Up to 254 files may be specified in the SELECT clauses within a source program.

4. The first 8 characters of file-name are used as the file link name unless the COBRUN operand LINK is specified when the program is compiled (for further information see the ASSIGN clause and the "COB1 User's Guide" [7]: COBRUN Operand). Since the link name cannot have more than 8 positions, file-name must be unique in these positions. If file-name is longer, it will be truncated for the link name. If it then ends with a hyphen in position 8, a warning is issued during compilation and the hyphen is converted into a number character (#).

5. If no FILE command is given for a file at object time, file-name is used as a BS2000 file name in a FILE macro. In this case, all characters in file-name (up to 30) are used.

ASSIGN CLAUSE

Function

The ASSIGN clause assigns an external device to a file of the COBOL program. One ASSIGN clause is required for each file in the program.

Format 1 is used for all files except sort-files

Format 2 see chapter "Sort-Merge Feature".

Format 1

```
ASSIGN TO [integer] [implementor-name-1] [implementor-name-2]...
```

Rules

1. The implementor-name designates a disk-storage unit and must be specified as follows:

   either

   a) `class-[device-][organization-] name`

   or

   b) `[[class-] device- [organization-]] name-1`

   (b) can only be used in connection with COBRUN LINK; see "COB1 User's Guide" [7].

   Class is a two-character field (optional in b only) that identifies a device class. Permitted: DA

   Device is an optional field that contains a symbolic device name, i.e. the type of device to which the file is to be assigned. Permitted: DISC 1). The device assigned is a disk storage unit.

   1) The entry DISC is used only for checking the validity of block/record length at compile time and does not affect the actual allocation of devices at object time. At object time, it is the user's responsibility to provide correct device assignments.

   Organization is a single-character field specifying the file organization. the following character may be used:

   | for files with indexed organization.

   Name is a six-character field that gives the symbolic name of the device to which the file is assigned. It must have the following format:

   SYSnnn (where nnn is an integer between 000 and 244)

The required LINK name is taken from the first 8 characters of the file-name given in the SELECT clause.

Name-1 in b) represents the required link name (see the COBRUN LINK description in the "COB1 User's Guide" [7]).

2. **Integer** indicates the number of devices assigned to the file. It is present only for reasons of compatibility, and is treated by the compiler as comments.

3. FOR MULTIPLE UNIT may be included whenever the number of disk-storage devices assigned might be less than the number of disk volumes of the file.

5

Programming considerations

1. Access can be sequential, random or dynamic/extended.

2. The file organization can also be given in the ORGANIZATION clause (q.v.).

ACCESS MODE CLAUSE

Function

The ACCESS MODE clause determines the manner in which the records of a file are to be accessed.

Format

```
                 ⎧SEQUENTIAL⎫
                 ⎪RANDOM    ⎪
ACCESS MODE IS   ⎨          ⎬
                 ⎪DYNAMIC   ⎪
                 ⎩EXTENDED  ⎭
```

Rules

1. SEQUENTIAL means that data records are read or written sequentially; that is, the next logical data record of the file is made available when executing a READ statement, or the next logical data record is placed on that file when executing a WRITE statement.

2. RANDOM means that data records are read or written on the basis of a key, i.e. access takes place using the RECORD KEY.

3. DYNAMIC means that direct and sequential access may be mixed.

4. EXTENDED and DYNAMIC mean the same thing.

5. When DYNAMIC is entered, a READ statement with or without INVALID KEY phrase indicates random access. A READ statement with NEXT phrase, or a READ statement with AT END phrase, indicates sequential access.

6. If the ACCESS MODE clause is not supplied, ACCESS IS SEQUENTIAL is assumed.

FILE STATUS CLAUSE

Function

The FILE STATUS clause specifies a data item whose contents are used for interrogating the status of input/output operations during processing. In addition, by specifying a further data item, the DMS code is made available.

Format

FILE STATUS IS data-name-1 [, data-name-2]

Rules

1. data-name-1 and data-name-2 may have one qualifier only.

2. data-name-1 and data-name-2 must be defined in the LINKAGE SECTION or WORKING-STORAGE-SECTION of the Data Division. Data-name-1 must be a two-byte numeric (DISPLAY only) or alphanumeric item. Data-name-2 must be a 6-byte group item in the form

```
0m          data-name-2.
0m+1        data-name-x PIC 9(3) COMP.
0m+1        data-name-y PIC X(4).
```

3. If the FILE STATUS clause is specified, the runtime system will

   a) move the I-O status to data-name-1, and

   b) if data-name-2 is specified, write data-name-2 as follows, depending on the value in data-name-1 and the DMS code:

      b1) If data-name-1 has the value 0, then data-name-2 is not rewritten, i.e. its contents are considered undefined.

      b2) If data-name-1 has a non-zero value and the DMS code is 0, then the left portion (first two bytes) of data-name-2 are written with 0 while the right position (the remaining four bytes) remain undefined. If, however, the DMS code is non-zero, the left portion of data-name-2 is given the value 128 and the right portion the DMS code (for further information see the DMS manuals).

   This takes place during execution of each OPEN, CLOSE, DELETE, READ, WRITE or REWRITE statement that references this file, and prior to execution of each corresponding USE procedure (see "I-O Status").

ORGANIZATION CLAUSE

Function

The ORGANIZATION clause defines the logical structure of a file.

Format

[ORGANIZATION IS] INDEXED

Rules

Unlike Standard COBOL where the file organization must be specified in this clause, it is also possible to enter the file organization optionally in the ASSIGN clause, using the organization field of the implementor-name (see "ASSIGN Clause"). The entry can also be made in both places, in which case it must be identical. If omitted altogether, ORGANIZATION IS SEQUENTIAL is assumed.

The entry can also be made in both places, in which case it must be identical.

If omitted altogether, ORGANIZATION IS SEQUENTIAL is assumed.

Programming consideration

File organization is defined at file construction time and is not, therefore, subject to modification.

RECORD KEY CLAUSE

The RECORD KEY clause defines the (prime) record key, and hence a means of accessing records in an indexed file.

Format

RECORD KEY IS data-name

Rules

1. data-name may have one qualifier only.

2. data-name must be a data defined within a record description entry. This record description entry must be assigned to the file-name to which the RECORD KEY clause is subordinate.

3. data-name may be any fixed-length item within the record. The item may be 1 through 255 bytes in length. Data-name may be qualified, but not indexed or subscripted.

4. If the file contains variable-length records, the record key must be contained in the first x character positions in the record, where x is the minimum record length for this file (see RECORD Clause).

5. Record key values must be unique for all records within the file.

6. The data description entry of data-name and the relative position within the record must be exactly as defined at file creation time.

7. If more than one record description entry exists in a file, data-name need only be defined in one of those entries. The same position specified by data-name relative to the start-of-record defines the implicit key position for all other record description entries in the file.

Programming considerations

1. An indexed file must be created sequentially. When creating an indexed file, the programmer must specify the RECORD KEY clause. During file creation, the object program uses the RECORD KEY clause to determine the contents and the location of the key in each record. Therefore, the programmer must set the RECORD KEY to a desired value before each WRITE statement. Since records are placed on the file in the order in which they are written, their keys must be specified in ascending collating sequence.

2. When retrieving or updating records sequentially, the programmer must specify the RECORD KEY clause. During sequential retrieval or update, data records marked for deletion are not retrieved.

3. For random retrieval and updating of records, the programmer must specify the RECORD KEY clause. He must also set the RECORD KEY to a desired value prior to each READ, WRITE or REWRITE statement.

4. The first byte of a record must not contain the value HIGH VALUE; otherwise, the record will be regarded as a dummy record by the system.

5. The user may reserve space for a new record by creating a dummy record while establishing a file.

RESERVE CLAUSE

Function

The RESERVE clause allows the user to modify the number of input/output areas (buffers) the compiled program will be allocated by the compiler.

Format

```
RESERVE { integer } { AREA  }
                    { AREAS }
```

Rules

1. integer must be an unsigned positive integer.
2. RESERVE AREA
   One input/output area is reserved.
3. RESERVE integer AREA
   The number of input/output areas indicated by integer are reserved.
4. RESERVE integer AREA
   The number of input/output areas specified under integer is reserved in addition to the input/output area reserved by default.

5.2.1.2 I-O-CONTROL Paragraph

Function

The I-O-CONTROL paragraph defines the events at whose occurrence restart points are to be established, and it specifies the memory area which is to shared by the various files.

Also, it indicates the location of files on multiple-file reels, and defines special input-output conditions.

Format

```
A    B      margin indication
|    |
v    v
I-O-CONTROL.

     [RERUN clause]...

     [SAME AREA clause]...

     [APPLY clause]... .
```

Rule

I-O-CONTROL must be written starting at margin A. All subsequent entries must be written in Area B.

For better readability, the clauses are described in the following subsections in alphabetical order.

RERUN CLAUSE

Function

A RERUN clause indicates where and when restart-point records are to be issued. A restart-point record describes the status of an object program at a specified point during program execution. It is produced automatically by the operating system upon the request of the object program and contains all information necessary to restart the program from that point. The RERUN clause controls such requests by the COBOL object program.

5

Format

```
RERUN [ON implementor-name]

        ⎧integer-1 RECORDS OF file-name-2⎫
  EVERY ⎨integer-2 CLOCK-UNITS           ⎬
        ⎩condition-name                  ⎭
```

Rules

1. implementor-name has the following format:
   SYSnnn.
   where: 000 ≤ nnn ≤ 244.
   nnn determines the processing mode.

   If nnn ≥ 200, the restart points are written alternately to two restart files. This makes it possible to restart from the two most recent restart points.

2. implementor-name designates the file to which the restart points are to be output.

3. Subject to the following restrictions, more than one RERUN clause may be specified for the same file-name-2:

   When specifying more than one integer-1 RECORDS phrase, the same file-name-2 must appear only once in them.

4. When implementor-name is specified, restart points are written as follows:

   a) If implementor-name is assigned to a file-name from the SELECT clause, the file involved must be a tape file. The effect on restart point output is the same as if the file-name had been specified directly in the RERUN clause.

   b) If implementor-name is not a file from a SELECT clause, a disk-storage file must be assigned at object time. Only in this case will restart point records be written to disk storage.

5. Restart points are requested when integer-1 RECORDS is entered. The restart points are written to the file specified by file-name-1 or implementor-name, respectively, whenever integer-1 records have been processed. File-name-2 may be either an input or output file with any organization or type of access; it must not be referenced in the USING/GIVING phrase or in an INPUT/OUTPUT procedure on Sort (see chapter "Sort-Merge Feature").

6. The CLOCK-UNITS phrase is treated by the compiler as comments.

7. The condition-name entry, too, is treated by the compiler as comments.

SAME AREA CLAUSE

Function

The SAME AREA clause indicates that two or more files are to share a specified input/output area during program execution.

Format 1 of the SAME AREA clause specifies all files except sort-files unless RECORD is supplied.

Format 2 see chapter "Sort-Merge Feature".

5

Format 1

SAME [RECORD] AREA FOR file-name-1 {file-name-2}...

Rules

1. More than one SAME AREA clause may be included in a program. In this case, the following must be observed:

   A specific file-name must not appear in more than one SAME AREA clause. The same is true of the SAME RECORD AREA clause.

   A specific file-name may concurrently appear in a SAME AREA clause and a SAME RECORD AREA clause. In this case, all file-names appearing in the SAME AREA clause must also appear in the SAME RECORD AREA clause. The SAME RECORD AREA clause may also contain other file-names that do not appear in the SAME AREA clause.

2. The SAME AREA clause indicates that the specified files (no sort-files) are to share the input/output areas assigned to them.

3. The SAME RECORD AREA clause indicates that the specified files are to share the same storage area for processing the current logical record.

   A logical record in the SAME RECORD AREA clause is considered a logical data record of all files opened for OUTPUT whose names are supplied in that SAME RECORD AREA clause. It is also a logical data record of the file (in this clause) from which the most recent input occurred. This is equivalent to an implicit overwriting of all data record areas, where the data records are aligned on the leftmost character position.

Programming considerations

1. If SAME AREA is used, only one file may be open at any given time.

2. If the RECORD phrase is used, all specified files may be open at the same time.

5.3 LANGUAGE ELEMENTS OF THE DATA DIVISION

5.3.1 FILE SECTION

Function

The FILE SECTION defines how the files are set up. Each file is defined by a file description entry and one or more record description entries. Record description entries are written immediately following the file description entry.

Format

A margin indication

FILE SECTION

[file description entry. {record description entry} ...]...

File description entries are discussed in the pages which follow. Record description entries are discussed in chapter 2 as well as in the pages which follow.

5.3.1.1 File Description (FD) Entry

Function

The file description (FD) entry specifies the physical setup, the identification, and the record names of a given file.

A file description entry must be written for each file to be processed by the program. The information contained in this entry generally pertains to the physical aspects of the file, that is, the description of the data as it appears on the input or output medium.

Format

```
A  B     margin indication
|  |
v  v
FD file-name

   [BLOCK CONTAINS clause]

   [RECORD clause]

   [RECORDING MODE clause]

   [LABEL RECORDS clause]

   [DATA RECORDS clause.]
```

Rules

1. The level indicator FD specifies the beginning of a file description entry and must precede the file-name.

2. The file-name must match the file-name given in a SELECT clause.

3. The clauses which follow the file-name are in some cases optional; The order in which they appear is immaterial.

4. The file description entry is terminated by a period.

5. If the LABEL RECORDS clause is omitted, LABEL RECORD IS STANDARD is assumed.

6. Table 5-2 provides a summary of the functions of the clauses used in file description entries.

   The formats and functions of these clauses are described on the following pages.

Programming considerations

The file description entry must be followed by one or more record description entries.

Clause	Function
BLOCK CONTAINS	Specifies block size (physical record size).
DATA RECORDS	Indicates the names of the data records in the file
LABEL RECORDS	Gives the names and values of the label records contained in the file. (LABEL RECORD IS STANDARD).
RECORD	Specifies logical record size.
RECORDING MODE clause	Specifies the format of the logical records

Table 5-2 Functions of file description clauses

BLOCK CONTAINS CLAUSE

Function

The BLOCK CONTAINS clause specifies the maximum size of a physical block.

Format

```
                                              ⎧CHARACTERS⎫
BLOCK CONTAINS [integer-1 TO] integer-2       ⎨          ⎬
                                              ⎩RECORDS   ⎭
```

Rules

1. integer-1 and integer-2 must both be at least 20 (for CHARACTERS only, since the minimum block length required by the system is 20) and at most 32763.

2. The CHARACTERS or RECORDS phrase indicates whether block length is to be specified as a multiple of characters or logical data records.

3. If neither the CHARACTERS nor the RECORDS phrase is used, CHARACTERS will be assumed.

4. integer-1 TO integer-2 indicates the number of characters or data records in a given block, depending on the omitted used (either CHARACTERS or RECORDS).

5. When only integer-2 is specified, it refers to the maximum size of the block. When integer-1 and integer-2 are specified, they refer to the minimum and maximum size of the block, respectively. However, the phrase "integer-1", when specified, is used only for documentation purposes and is treated as comments by the compiler.

   The meaning of the **maximum block size** specified by this clause is the following:

   The blocks of the file may not be longer but may be shorter than the specified length, as is frequently the case with unblocked or blocked data records whose length is variable.

6. When the CHARACTERS phrase is used, the block size is specified in terms of the number of characters contained within the block, regardless of the types of characters used to represent the data items within the block. In this case, both integer-1 and integer-2 must include slack bytes and four characters for the record length field of each record of the block. The compiler will add four characters for the block length field.

7. When the CHARACTERS phrase is used, and only integer-2 is specified, integer-2 indicates the length of the physical block. When both integer-1 and integer-2 are specified, they refer to the minimum or maximum physical block length, respectively.

8. If the RECORDS phrase is used, the block size is specified as the number of logical records. In this case, the compiler calculates the block size by multiplying the number of characters in the maximum record size by the value specified in integer-2, plus 4 characters for the block size field in the case of variable-length records.

9. When the BLOCK CONTAINS clause is omitted, the compiler assumes that the records are not blocked, i.e. BLOCK CONTAINS 1 RECORDS is assumed. Consequently, the BLOCK CONTAINS clause may be omitted when **all** blocks of the file contain one, and only one, data record.

Programming considerations

1. Table 5-3 shows how the compiler calculates block sizes in terms of characters, and which additional specifications are contained in the BLOCK CONTAINS clause and the RECORDING MODE clause.

   Legend for Table 5-3:

   F = fixed-length records
   V = variable-length records
   BL = block length
   SL = record length
   SL_{max} = maximum record length
   BLF = block length field (has the value 4)
   SLF = record length field (has the value 4)
   n = integer

Mode option of RECORDING MODE clause	BLOCK CONTAINS [integer-1 TO] integer-2	
	CHARACTERS	RECORDS
F	BL = integer-2 (integer-2 = n*SL)	BL = integer-2*SL
V	BL = integer-2+BLF	BL = integer-2*(SLmax+↓SLF↓)+BLF

Table 5-3 Calculation of the maximum block size

2. The CHARACTERS phrase should be used if the RECORDS phrase would imply a too inaccurate record size. For example, assume that the programmer describes a mode V Record of 100 characters, and that he writes a block consisting of a 50-character record followed by three 100-character records. If he specifies the clause BLOCK CONTAINS 4 RECORDS, the compiler will assume that the block length is 420 characters (i.e. 4 x (100 + 4) +4).

   However, since each block requires only 366 + 4 characters (i.e. (50+4) + 3 x (100+4) +4), the programmer should specify the following clause:

```
BLOCK CONTAINS 366 CHARACTERS.
```

DATA RECORDS CLAUSE

Function

The DATA RECORDS clause serves only as documentation. It specifies the names of the data records in a file.

Format

```
     ┌RECORD   IS ┐
DATA ┤            ├ data-name-1 [data-name-2]...
     └RECORDS  ARE┘
```

Rules

1. data-name-1, data-name-2 etc. are the names of data records. Each of these records must be preceded by the level number 01 in the file description entry.

2. The presence of more than one data-name indicates that the file contains more than one type of data record. These records may be of differing sizes, different formats, etc. The order in which they are listed is not significant.

LABEL RECORDS CLAUSE

Function

The LABEL RECORDS clause specifies whether labels are present; and if labels are present, this clause identifies them.

Format

$$\underline{\text{LABEL}} \left\{ \begin{array}{l} \underline{\text{RECORD}}\ \text{IS} \\ \underline{\text{RECORDS}}\ \text{ARE} \end{array} \right\} \underline{\text{STANDARD}}$$

Rule

The STANDARD phrase specifies that labels are present for the file and that these labels are in accordance with system standards (see [2]).

RECORD CLAUSE

Function

The RECORD clause defines the length of the records in a file.

Format 1 Indicates fixed-length records by specifying the number of character positions in a record.

Format 2 Indicates variable-length records whose record size must lie within a certain range.

Format 3 Indicates variable-length records, specifying the minimum and maximum number of character positions.

The length of each data record is defined by its record description entry. If the RECORD clause is specified, the record length is compared with the entries in this clause (see also RECORDING MODE clause).

Format 1

```
RECORD CONTAINS integer-1 CHARACTERS
```

Rules

1. The number of character positions of each record description entry must be equal to integer-1.
2. integer-1 must be at least 1 and can be as large as 32767.

Format 2

```
RECORD IS VARYING IN SIZE [[FROM integer-2] [TO integer-3] CHARACTERS]
    [DEPENDING ON data-name-1]
```

Rules

1. In any record description entry, it is impermissible for the length specification to be less than integer-2 or more than integer-3.
2. integer-3 must be greater than integer-2.
3. integer-2 must be at least 1, while integer-3 can be up to 32763.
4. data-name-1 must be described as an unsigned integer data item in the WORKING-STORAGE or LINKAGE section.

5. The length of a record is determined by the sum of the character positions of all its constituent data items and the slack bytes created by the compiler. Data description entries described with the REDEFINES or RENAMES clause are not counted when calculating the length. If the record contains a table, the minimum or maximum number of table items is taken into account when calculating the length (for further details see SYNCHRONIZED and USAGE clauses as well as "Data Alignment").

6. If integer-2 is omitted, the compiler assumes that the length of the shortest record in the file is that portion of the record description entry which contains the record key, from the 01-level up to and including the record key.

7. If integer-3 is omitted, the length of the longest record in the record description entry for this file is assumed. This number also provides the block size.

8. If data-name-1 is specified, the number of character positions in the record must be moved to data-name-1 before a REWRITE or WRITE statement is executed for this file.

9. If data-name-1 is specified, its contents remain unchanged when a DELETE, REWRITE, START or WRITE statement is executed or when a READ statement terminates abnormally.

10. During execution of a REWRITE or WRITE statement, the record length is determined as follows:

    a) By the contents of data-name-1, if specified.

    b) By the number of character positions in the record if data-name-1 is omitted and the record description entry has no OCCURS clause with DEPENDING ON phrase.

    c) By the fixed portion of the record description entry (i.e. all data description entries without OCCURS DEPENDING ON) and by the number of recurrences of the table items during execution when data-name-1 is omitted and the record description entry has no OCCURS clause with DEPENDING ON phrase.

11. If data-name-1 is specified and a READ statement has been successfully executed, data-name-1 contains the number of character positions of the record just read. This number is, however, not greater than integer-3, but it may be larger than the largest record description entry.

12. If INTO is specified in a READ statement, the number of character positions in the current record, which is the sending field in the implicit MOVE, is determined as follows:

    a) By the contents of data-name-1, if specified.

    b) By the value that would be transferred if data-name-1 had been specified.

Format 3

<u>RECORD</u> CONTAINS integer-4 <u>TO</u> integer-5 CHARACTERS

Rules

1. integer-4 describes the number of character positions in the shortest record, integer-5 the number in the longest.
   For each record, however, the size defined in the record description entry applies.

2. The length of a record is determined by the sum of the character positions in all elementary items and by the slack bytes created by the compiler. Data description entries described with the REDEFINES or RENAMES clause are not counted in the length calculation. If the record contains a table, the minimum or maximum number of table items is taken into account in the length calculation. (For further information see the SYNCHRONIZED and USAGE clause as well as "Data Alignment".)

3. integer-5 must not be more than 32763.

RECORDING MODE CLAUSE

Function

The RECORDING MODE clause specified the format of the logical records in the file.

Format

RECORDING MODE IS mode

Rules

1. mode indicates a record format, and may be F or V.

2. F

   indicates fixed-length records. It can only be specified if all logical records in the file have the same length. When mode F is specified, the following rules apply:

   a) No record description entry in the file may contain an OCCURS clause with DEPENDING ON phrase.

   b) Blocks may contain more than one record; in this case, a fixed number of records per block is implied.

   c) If more than one record description is given for a file description (FD) entry, all records in the file must have the same length.

   d) In F-mode blocks and data records, no block-length fields (BLF) and record length fields (SLF) are used.

3. V

   indicates variable-length records, and may be used to describe either fixed-length or variable-length records. When V mode is specified, the following rules apply:

   a) Blocks may contain more than one record; however, all of the logical records specified for a block must be wholly contained within the block.

   b) Each V-mode record is preceded by a control field containing the length of the logical record (record length field, SLF). A block of variable length records also contains a block descriptor control field (block length field, BLF). These control fields must not be described in Data Division record descriptions, since the compiler automatically provides for them. These fields are not available to the user.

5

Programming considerations

If the RECORDING MODE clause is not explicitly written for an FD entry, the recording mode of the file is determined by the RECORD clause, as shown in Table 5-4.

a) If the RECORD clause is not specified, the recording mode assumed for the file is V (variable).

b) If the RECORD CONTAINS clause specifies only one number (i.e. RECORD CONTAINS integer-1 CHARACTERS), then the recording mode assumed for the file is F (fixed).

c) If the RECORD CONTAINS clause specifies two numbers (i.e. RECORD CONTAINS integer-4 TO integer-5 CHARACTERS), then the recording mode assumed for the file is V (variable).

d) If the RECORD IS VARYING IN SIZE clause is specified, it is always assumed that records are variable-length.

Mode option of RECORDING MODE clause	Entry in the RECORD clause RECORD CONTAINS clause not written	integer-1 CHARACTERS	integer-4 TO integer-5 CHARACTERS	RECORD IS VARYING IN SIZE
None specified	V	F	V	V
F	F	F	V 1)	V 1)
V	V	V 1)	V	V

Table 5-4 Relationships between the RECORD CONTAINS clause and the RECORDING MODE clause

1) The two clauses are contradictory; V is assumed.

5.4 LANGUAGE ELEMENTS OF THE PROCEDURE DIVISION

5.4.1 INVALID KEY Condition

The INVALID KEY condition may occur after the execution of a START, READ, WRITE, REWRITE or DELETE statement. Details about the occurrence of this condition are described under each of these statements.

If an INVALID KEY condition was encountered, the File Control Processor performs the following actions in the specified order.

1. A value is entered into the FILE STATUS data item, if specified for this file, to indicate the INVALID KEY condition (see "FILE STATUS Clause").

2. If the statement causing an invalid key condition includes an INVALID KEY phrase, then the imperative statement following INVALID KEY is executed. Any USE procedure supplied for that file will not be executed.

3. If no INVALID KEY phrase is present, but a user procedure exists for that file, the USE procedure is executed. This may be an explicitly or implicitly declared user procedure.

4. If neither the INVALID KEY phrase nor a USE procedure is present for this file, the invalid key condition causes the program to terminate abnormally.

If an invalid key condition occurs, the execution of the input/output statement which recognized the condition is unsuccessful, and the file is in the same condition as it was before this input/output statement was executed.

5.4.2 Input/Output Statements

In COBOL, input and output is record-oriented. Thus the READ, WRITE, DELETE and REWRITE statements process records. The COBOL user is therefore only concerned with the processing of single records. The following operations are performed automatically: moving data into input/output areas (buffers) and/or internal storage, validity checking, and error correction (where feasible).

Summary

Statement	Function
CLOSE	Terminates processing of a file.
DELETE	Removes a data record.
OPEN	Opens a file for processing.
READ	Makes a record available from an input or input/output file.
REWRITE	Replaces a data record on a disk-storage file by a specified data record.
START	Positions within a file.
USE	In addition to input/output statements, the USE statement may be supplied to indicate label and error handling procedures (see "Declaratives Subdivision of the Procedure Division").
WRITE	Releases a record to an input/output or output file.

CLOSE STATEMENT

Function

The CLOSE statement terminates the processing of files, with optional rewind and/or lock where applicable.

Format

```
CLOSE file-name-1 [WITH LOCK]
     [file-name-2] [WITH LOCK]]...
```

Rules

1. file-name-1, file-name-2,... specify files on which the CLOSE statement is to operate.

2. file-name-1, file-name-2,... must be described in a file description (FD) entry in the Data Division of the program.

3. The files referenced in the CLOSE statement need not all have the same organization or access.

4. A CLOSE statement may be executed only on an open file.

5. The following paragraphs describe the meanings of the various CLOSE statements in random access file processing.

   The results of executing each CLOSE phrase are summarized in table 5-5. The definitions of the symbols used therein are given below.

   K - Standard Close File

   The standard system close procedures are implemented.

   L - Standard File Lock

   The compiler ensures that the file cannot be reopened while the program is running.

CLOSE Phrase	File Type (random access)	
	Single-volume	Multi-volume
CLOSE	K	K
CLOSE WITH LOCK	K, L	K, L

Table 5-5 Relationship of types of random-access files and CLOSE statement options.

6. After execution of an CLOSE statement, the contents of the data-name specified in the FILE STATUS clause will be updated (see also "FILE STATUS Clause").

Programming considerations

1. After successful execution of a CLOSE statement, the record area assigned to the file is no longer accessible, whereas availability of the record area after unsuccessful execution of a CLOSE statement is unpredictable.

2. If a file remains open after execution of a STOP RUN statement, it will be closed, like all files left open after task termination.

3. If a CLOSE statement was executed for a file, the next input/output statement to be executed must be an OPEN statement.

4. The LOCK phrase signifies that the file cannot be reopened within the same program run.

DELETE STATEMENT

Function

A DELETE statement logically removes a data record from a disk-storage file.

Format

DELETE file-name RECORD [INVALID KEY imperative-statement]

Rules

1. file-name must be the name of a disk-storage file.

2. For a file whose access mode is sequential, the INVALID KEY phrase of the DELETE statement must not be supplied.

3. For a file whose access mode is other than sequential, the INVALID KEY phrase is mandatory if no appropriate USE procedure is supplied.

4. For a file whose access mode is sequential, the last input/output statement for the associated file, which preceded the DELETE statement must have been a successfully completed READ statement. The RELATIVE KEY may not be changed between reading and deletion. Execution of the DELETE statement causes the data record read by the preceding READ statement to be deleted from that file.

5. For a file with random or dynamic/extended access, the input/output system deletes the record identified by the contents of the data item specified in the RELATIVE KEY clause of the file. If the record referenced by the key does not exist on the file, an INVALID KEY condition occurs (see "INVALID KEY Condition").

6. After execution of the DELETE statement, the contents of the data item specified in the FILE STATUS clause for the file is updated (see "FILE STATUS Clause").

Programming considerations

1. The file referenced in the DELETE statement must be in I-O open mode during the execution of this statement (see also "OPEN Statement").

2. After successful execution of the DELETE statement, the identified data record is deleted from the file.

3. Execution of a DELETE statement does not affect the contents of the data record area associated with the file.

OPEN STATEMENT

Function

The OPEN statement opens files for processing.

Format

```
      ┌INPUT file-name-1 [file-name-2]...  ┐
OPEN  ┤OUTPUT file-name-3 [file-name-4]... ├...
      └I-O file-name-5 [file-name-6]...    ┘
```

5

Rules

1. file-name-1, file-name-2, file-name-3,... must be defined by a file description (FD) entry in the Data Division.

2. At least one of the INPUT, OUTPUT or I-O phrases must be used. If two or more such phrases are used, they may appear in any order. Two or more file-names may be supplied for one option; however, the same file-name must not appear more than once in a given OPEN statement.

3. The files specified in a given OPEN statement may have different organizations or access modes.

4. After successful execution of an OPEN statement, the file specified by file-name is available in the open mode.

5. After successful execution of an OPEN statement, the associated record area is available to the program.

6. Before successful execution of an OPEN statement for a file, no statement may be executed that would either explicitly or implicitly reference that file.

7. INPUT means that the file is to be processed as an input file (input mode).

8. OUTPUT indicates that the file is to be processed as an output file (output mode).

9. I-O indicates that input and output operations (i.e. reading, writing, and updating operations) are to be performed on the file. Since this phrase implies the existence of the file, it cannot be used if the file is being initially created (update mode).

10. If the FILE STATUS clause is specified, the contents of the data-name given in the FILE STATUS clause are updated following execution of an OPEN statement (see also FILE STATUS Clause).

Programming considerations

1. All of the INPUT, OUTPUT or I-O phrases may be used, within the same program, in different OPEN statements for a given file. Before executing another OPEN statement after the logically first one for the same file in a program, a CLOSE statement must have been performed; at this time, LOCK may not appear in in the CLOSE statement.

2. The open mode specified in the OPEN statement is ignored if a FILE command with OPEN operand is issued for the same file before the program starts.
   If the OPEN operands given in the OPEN statement and the FILE command conflict, the OPEN statement will be unsuccessful (I-O status 39). For more precise information see the "COB1 User's Guide" [7].

3. Table 5-6 lists the statement permissible in OPEN mode (x = "permitted").

ACCESS clause	Statement	OPEN mode		
		INPUT	OUTPUT	I-O
SEQUENTIAL	READ	X		X
	WRITE		X	
	REWRITE			X
	START	X		X
	DELETE			X
RANDOM	READ	X		X
	WRITE		X	X
	REWRITE			X
	START			
	DELETE			X
DYNAMIC	READ	X		X
EXTENDED	WRITE		X	X
	REWRITE			X
	START	X		X
	DELETE			X

Table 5-6 Valid input-output statements in each OPEN mode

READ STATEMENT

Function

The READ statement places at the disposal of the program:

- the next data record in the file when access is sequential
- a particular data record in a disk file (specified by the key in the data record) when access is random.

Format 1 is used for all files with sequential or dynamic/extended access in order to process records sequentially.

Format 2 is used for all files with random or dynamic/extended access in order to process records randomly (RECORD KEY).

The addendum WITH NO LOCK appearing in both formats is effective when simultaneous processing is used; it is described in the "COB1 User's Guide" [7] (see under "SHARUPD").

Format 1

```
READ file-name [WITH NO LOCK] [NEXT] RECORD [INTO identifier]
                              [AT END imperative-statement]
```

Rules for format 1

1. file-name must appear in a file description entry in the Data Division of the program.

2. NEXT must be specified when data records are to be read sequentially from a file using dynamic/extended access mode.

3. AT END must be specified when there is no USE procedure for the file.

4. The READ statement makes available to the program a record from the file specified by file-name, unless an exceptional condition exists. Here, the data record is read into the input area which was specified by the record description entry in the file description. It remains available there until the next input/output statement is executed for that file.

5. A READ statement with INTO phrase specified causes the following to happen:

   a) The data record is read into the input area (as described under 4 above).

   b) This data record is then moved from the input area to the areas specified by identifier (implicit MOVE). The MOVE operation takes place according to the rules for MOVE statements without CORRESPONDING phrase. After the READ statement with INTO phrase has been successfully executed, the data record is available both in the input area and in the area specified by the identifier. The length of the sending field is determined by the length of the data record which was read in (see "RECORD Clause"). If the file description entry contains a RECORD clause with VARYING phrase, the implicit MOVE is a group MOVE operation.

      If the operation described in a) was unsuccessful, no MOVE takes place.

      The index for identifier is calculated after the operation described in a) is executed but immediately prior to the implicit MOVE.

6. When executed, a READ statement causes the data item specified in the FILE-STATUS clause for the relevant file to be updated (I-O Status; see also "FILE-STATUS Clause").

7. Following an unsuccessful attempt to perform a READ statement, the contents of the record area associated with the file as well as the position within the file are unpredictable.

8. If the number of character positions in a read data record is greater than the largest record description entry, the data record will be truncated accordingly; if it is smaller than the smallest record description entry, the contents to the right of the last valid character will be unpredictable (for the length of a record description see "RECORD Clause"). The READ operation is successful in both cases, but a FILE STATUS will be set indicating the occurrence of a record length conflict.

Programming considerations for format 1

1. An OPEN statement with the INPUT or I-O phrase must be executed for a file before the READ statement can be executed.

2. When a file contains more than one logical record type, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. Only the information in the current record is accessible. Thus, no statement in the Procedure Division can refer to a different logical record. Similarly, no reference is permitted to the nth occurrence of data in a data record that occurs fewer than n times (see the description of the DEPENDING ON phrase, under the OCCURS clause). If either type of reference is attempted, the results will be unpredictable.

3. If a READ statement is executed, and there is no further data record on the file, then an end-of-file condition occurs, and the execution of that READ statement is considered unsuccessful (see also "FILE STATUS Clause").

4. If the end-of-file condition is encountered, the following steps are performed, in the order given below:

   a) The contents of the data item from the FILE STATUS clause, if specified for the file, is supplied with a value indicating the AT END condition (see "FILE STATUS Clause").

   b) If AT END is specified in the READ statement, control is transferred to the imperative statement following the AT END phrase. Any USE statement declared for that file will **not** be executed.

   c) If the AT END phrase is omitted from the READ statement, either an explicit or implicit USE procedure must be declared for the file. When the AT END condition occurs, that USE procedure will then be executed.

   When the AT END condition occurs, execution of the input/output statement must be considered to be unsuccessful.

5. After occurrence of the AT END condition, no format-1 READ statement must be issued for the file before one of the following steps is taken:

   a) A successful CLOSE statement, followed by a successful OPEN statement for that file.

   b) A successful START statement for that file.

   c) A successful format-2 READ statement for that file.

Format 2

```
READ file-name [WITH NO LOCK] RECORD [INTO identifier]
                              [INVALID KEY imperative-statement]
```

Rules for format 2

1. file-name must be defined by a file description entry in the Data Division.

2. If no USE procedure is declared for the file, the INVALID KEY condition must be supplied in the READ statement.

3. The READ statement makes available to the program a record from the file specified by file-name, unless an exceptional condition exists. Here, the data record is read into the input area which was specified by the record description entry in the file description. It remains available there until the next input/output statement is executed for that file.

4. A READ statement with INTO phrase specified causes the following to happen:

   a) The data record is read into the input area (as described in 3 above).
   b) This data record is then moved from the input area to the area specified by identifier (implicit MOVE). The MOVE operation takes place according to the rules for MOVE statements without CORRESPONDING phrase. After the READ statement with INTO phrase has been successfully executed, the data record is available both in the input area and in the area specified by identifier. The length of the sending field is determined by the length of the data record being read.

   If the operation described in a) was unsuccessful, no move takes place.

   The index for identifier is calculated after the operation described in a) is executed but immediately prior to the implicit MOVE.

5. Execution of a READ statement causes the contents of the data item specified in the FILE STATUS clause for the related file to be updated (I-O status; see also "FILE STATUS Clause").

6. Following an unsuccessful attempt to perform a READ statement, the contents of the input area are associated with the file as well as the position within the file are unpredictable.

7. If a file is accessed in random or dynamic/extended mode, the contents of the data item associated with the RECORD KEY must be set to the required value before the READ statement is executed (see "RECORD KEY Clause").

8. When a file is being read randomly, a search is made for a data record whose relative record number is equal to the value of the RECORD KEY item of that file. If no such data record exists on the file, an INVALID KEY condition occurs (see also "INVALID KEY Condition").

Programming considerations for Format 2

1. An OPEN statement with the INPUT or I-O phrase must be executed for a file before the READ statement can be executed.

2. When a file contains more than one logical record type, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. Only the information in the current record is accessible. Thus, no statement in the Procedure Division can refer to a different logical record. Similarly, no reference is permitted to the nth occurrence of data in a data record that occurs fewer than n times (see the description of the DEPENDING ON phrase, under the OCCURS clause). If either type of reference is attempted, the results will be unpredictable.

REWRITE STATEMENT

Function

The REWRITE statement replaces a logical record on a disk-storage file.

Format

```
REWRITE record-name [FROM identifier]
       [INVALID KEY    imperative-statement]
```

Rules

1. record-name and identifier must not refer to the same storage area.

2. record-name must be defined in a file description (FD) entry in the Data Division of the program.

3. The INVALID KEY phrase must be specified for a REWRITE statement, unless an applicable USE procedure was declared.

4. record-name identifies the record to be replaced.

5. Execution of a REWRITE statement with the FROM phrase specified corresponds to the following statements:

   MOVE identifier TO record-name
   REWRITE record-name.

   The contents of the storage area described by record-name will not effect execution of the REWRITE statement until the implicit MOVE statement has been performed.

   Therefore, when using the FROM phrase, data is transferred from identifier to record-name and is then released to the appropriate file. Identifier may be used for referencing any data outside the current file description entry.

   Transfer of data through the implicit MOVE statement takes place in accordance with the rules for a MOVE statement without the CORRESPONDING phrase. After the REWRITE statement is executed, the contents of the record is still available in the area specified by the "identifier" although it is no longer available in the area specified by record-name.

6. The following rules apply to all files whose access mode is sequential:

   a) A REWRITE statement must be preceded by a successful READ statement as the last input/output statement for the associated file.

   b) Execution of the REWRITE statement causes the record accessed by the preceding READ statement to be replaced on that file.

7. For any file with an access mode other than sequential, the RECORD KEY must be supplied with an appropriate value before the REWRITE statement is executed.

8. Indexed files with a sequential access mode are, again, subject to Rule 6. It should be noted additionally that the value of the data item specified by the RECORD KEY clause for the file must not be modified between the READ and the REWRITE statements.

9. For indexed files whose access mode is random or dynamic/extended, the REWRITE statement must not be preceded by a READ statement. In this case, the REWRITE statement will replace the data record according to the contents of the data item specified by RECORD KEY. That data item must have been set to the required value before the REWRITE statement is executed.

10. If the file does not contain a data item corresponding to the contents of the RELATIVE KEY item, the INVALID KEY condition occurs (see "INVALID KEY Condition").

11. The data record rewritten by a successful REWRITE statement is no longer accessible in the record area; an exception to this rule is the use of the SAME RECORD AREA clause. In this case, the data record is available to all other files specified in the SAME RECORD AREA clause as well as to the current file.

12. Execution of a REWRITE statement causes the contents of the data item that was specified in the FILE STATUS clause of the related file description entry to be updated (see also "FILE STATUS Clause").

13. The number of character positions in the record indicated by record-name cannot be greater than the largest record description entry for the associated file or less than the number of character positions permitted by the associated RECORD clause. If one of these errors occurs, the WRITE statement will terminate abnormally. The update operation will not take place, and the contents of the record area will remain unchanged. The I-O status of the file associated with record-name will be set to a value indicating the cause of the condition (see "I-O Status").

Programming considerations

1. The file associated with record-name must be a disk-storage file and must have been opened I-O at the time the REWRITE statement is executed.

2. The data record addressed by the data record name must be of the same length as the data record to be replaced on the file.

3. When a data record is being rewritten, the first byte of the record must not contain the value X'FF' as this would effect an illogical deletion of the record. This method of deleting a record is retained in COB1 though the DELETE verb is supplied for issuing a delete statement.

START STATEMENT

Function

The START statement defines the logical starting point, within a file, for subsequent sequential read operations.

Format

```
                                        ┌ ┌IS EQUAL TO                  ┐            ┐
                                        │ │IS =                         │            │
                                        │ │IS GREATER THAN              │            │
                                        │ │IS >                         │            │
                                        │ │IS NOT LESS THAN             │            │
START file-name [WITH NO LOCK]          │ KEY ┤IS NOT <                 ├ data-name  │
                                        │ │IS LESS THAN                 │            │
                                        │ │IS <                         │            │
                                        │ │IS LESS THAN OR EQUAL TO     │            │
                                        │ │IS NOT GREATER THAN          │            │
                                        └ └IS NOT >                     ┘            ┘
      [INVALID KEY imperative-statement]
```

The addendum WITH NO LOCK is effective when simultaneous processing is used; it is described in the "COB1 User's Guide" [7] (see under "SHARUPD").

Rules

1. Relational operators, when specified, are indispensable separators.

2. file-name must refer to a file with sequential or dynamic/extended access mode.

3. data-name may be qualified.

4. The INVALID KEY phrase must be present if no applicable USE procedure has been declared for that file.

5. data-name can be the name of the RECORD KEY data item declared for that file or, in the case of OPEN mode I-O only, it can be an alphanumerically described data item which is subordinate to the RECORD KEY item. In this case the first characters of both items must be identical.

6. The type of comparison is specified by the relational operator in the KEY phrase. The logical key of each record in the file specified by data-name is compared with the contents of the data item referenced by data-name (RECORD KEY). If the length of data-name differs from that of the RECORD KEY item, the comparison is performed as though the larger data item had been truncated from the right to the length of the smaller item. All remaining rules governing the comparison of nonnumeric operands then apply, with any specified PROGRAM COLLATING SEQUENCE being ignored. All key comparisons thus take place on the basis of the EBCDIC collating sequence, i.e. as though no PROGRAM COLLATING SEQUENCE or PROGRAM COLLATING SEQUENCE IS NATIVE had been specified.

   a) For the relational operators EQUAL, GREATER, GREATER OR EQUAL, NOT LESS and their equivalents, the current record pointer is positioned to the first record currently existing in the file whose key satisfies the comparison.

   b) For the relational operators LESS, LESS OR EQUAL, NOT GREATER and their equivalents, the record pointer is positioned to the last record whose key satisfies the comparison.

   c) If the relational condition is not satisfied for any record in the file, an INVALID-KEY condition occurs, and the START statement is prematurely terminated (see "INVALID-KEY Condition").

7. If the KEY phrase is omitted from the START statement, the comparison described under 6) uses the data item referenced in the RECORD KEY clause associated with file-name, and the relational operator "=" will be assumed.

8. After execution of a START statement, the contents of the FILE STATUS data item of that file (if specified) is updated (see also "FILE STATUS Clause").

9. The file specified by file-name must have been opened by INPUT or I-O at the time the START statement is executed (see "OPEN Statement").

USE STATEMENT

Function

The USE statement introduces declarative procedures and defines the condition for their execution. The USE statement itself, however, is not executed.

Format 1 see "Sequential File Organization".

Format 2 specifies procedures to be executed if an input/output error occurs in a file.

Format 2

```
                    {ERROR    }
USE AFTER STANDARD  {         }  PROCEDURE
                    {EXCEPTION}

          {file-name-1 [file-name-2]...}
          {INPUT                       }
      ON  {                            }
          {OUTPUT                      }
          {I-O                         }
```

Rules

1. The ERROR and EXCEPTION phrases are equivalent and may be used optionally.

2. file-name must be defined in a file description (FD) entry of the program's Data Division.

3. When the file-name phrase is specified, the error handling procedures are executed only for the files named. No further USE procedures are run for these files.

4. The specified procedures are executed by the system when an input/output error occurs during the execution of an input/output statement for a file.

5. Before execution of the user error routine, the standard system error routines for input/output error handling are executed.

6. After a USE procedure is executed, control passes to the calling routine.

7. INPUT indicates that the specified procedures are executed only for files opened in input mode (OPEN statement with the INPUT option specified).

8. OUTPUT indicates that the specified procedures are executed only for files opened in output mode (OPEN statement with the OUTPUT phrase specified).

9. I-O indicates that the specified procedures are executed only for files opened in update mode (OPEN statement with the I-O option specified).

Programming considerations

1. The USE procedures are run when:

   a) an INVALID KEY or AT END condition occurs, provided that the input/output statement where this occurrence takes place does not include an INVALID KEY or AT END phrase.

   b) a severe error occurs (FILE STATUS CODE ≥ 30).

   If no corresponding USE procedure exists for the file when a) or b) occurs, the program will abort.

2. Files referenced either implicitly (INPUT, OUTPUT and I-O) or explicitly (file-name-1, file-name-2,...) in the USE statement need not have the same organization and access mode.

3. Within a USE procedure, there must be no reference to any non-declarative procedures excepting the PERFORM statement.

   Reference to procedure names which are subordinate to a USE statement may be made from another procedure, or from a non-declarative procedure, only by using a PERFORM statement.

4. File-name may not appear in more than one USE statement.

Example 5-1 (format 2)

Environment Division statements::

```
     SELECT MASTER-FILE ASSIGN DA-DISC-R-SYS001
            RECORD KEY ISAMKEY
            FILE STATUS INDICATOR.
 .
 .
 .
```

Data Division entries:

```
  FILE SECTION.
  FD  MASTER-FILE LABEL RECORD STANDARD.
  01  RECORD-FORMAT
      03 ISAMKEY                PIC 9(8).
      03 CONTENTS               PIC X(72).
  WORKING-STORAGE SECTION.
  77  INDICATOR                 PIC 99.
  77  CLOSE-INDICATOR           PIC X VALUE "0".
  .
  .
  .
```

Procedure Division statements:

```
PROCEDURE DIVISION.
DECLARATIVES.
INPUT-ERROR SECTION.
    USE AFTER ERROR PROCEDURE ON MASTER-FILE.
STATUS-QUERY.
    IF INDICATOR NOT < 30
        GO TO UNRECOV-ERROR.
END-OF-FILE.
    IF INDICATOR = 10
        DISPLAY "END OF MASTER-FILE ENCOUNTERED."
        GO TO SPOOL-BACK.
OUTPUT-DUPKEY.
    IF INDICATOR = 21
        DISPLAY "RECORD WITH KEY" ISAMKEY
        "ALREADY EXISTS"
        "OR NOT IN ASCENDING ORDER"
        GO TO SPOOL-BACK.
I-O-DUPKEY.
    IF INDICATOR = 22
        DISPLAY "RECORD WITH KEY" ISAMKEY
        "ALREADY EXISTS"
        GO TO SPOOL-BACK.
NON-EXISTENT.
    IF INDICATOR = 23
        DISPLAY "RECORD WITH KEY" ISAMKEY
        "NON-EXISTENT"
        GO TO SPOOL-BACK.
UNRECOV-ERROR.
    DISPLAY "UNRECOVERABLE ERROR ("INDICATOR")
        "FOR MASTER FILE".
    IF CLOSE-INDICATOR = "1"
        CLOSE MASTER-FILE.
    DISPLAY "PROGRAM TERMINATED ABNORMALLY".
    STOP RUN.
SPOOL-BACK.
    EXIT.
END DECLARATIVES.
STARTUP.
    OPEN INPUT MASTER-FILE.
    MOVE "1" TO CLOSE-INDICATOR.
    .
    .
    .
    MOVE "0" TO CLOSE-INDICATOR.
    CLOSE MASTER-FILE.
.
.
.
```

5

WRITE STATEMENT

Function

The WRITE statement causes a data record to be released to an input/output or an output file.

Format 1 see "Sequential File Organization".

Format 2 of the WRITE statement must be used for disk-storage files with any organization other than sequential.

Format 2

```
WRITE record-name [FROM identifier-1]
     [INVALID KEY imperative-statement]
```

Rules

1. record-name and identifier-1 must not reference the same storage area.

2. record-name is the name of a data record in the FILE SECTION and must not be qualified by a file-name.

3. record-name must not be part of a sort-file.

4. The INVALID KEY phrase must be specified for a file unless a corresponding USE procedure was declared.

5. The data record supplied by a WRITE statement is no longer available in the record area, unless the file related to the data record was specified in a SAME RECORD AREA clause or the execution of the WRITE statement was abnormally terminated as unsuccessful because of the occurrence of an INVALID KEY condition. The record is also available to those files which were referenced in a SAME RECORD AREA clause together with the specified file.

6. Execution of a WRITE statement with the FROM phrase is equivalent to the following statements:

   MOVE identifier-1 TO record-name.
   WRITE record-name.

   Data is transferred according to the rules for a MOVE statement without the CORRESPONDING phrase.

   The contents of the record area before execution of the implicit MOVE statement has no effect on the execution of the WRITE statement.

   After the WRITE statement is successfully executed, the information is still available in the area referred to by the identifier; however, as pointed out under Rule 5, this is not necessarily applicable to the data record area.

7. If the end-of-volume condition is encountered on a multi-volume output file for disk-storage and sequential access mode, then execution of the WRITE statement causes the following steps to be performed:

   a) The standard ending volume label procedures are performed.

   b) A volume switch takes place.

   c) The standard beginning volume label procedures are performed.

8. After execution of a WRITE statement, the value of the data item of any FILE STATUS clause existing for that file is updated (see "FILE STATUS Clause".

9. The number of character positions in the record indicated by record-name cannot be greater than the largest record description entry for the associated file or less than the number of character positions permitted by the associated RECORD clause. If one of these errors occurs, the WRITE statement will terminate abnormally. The update operation will not take place, and the contents of the record area will remain unchanged. The I-O status of the file associated with record-name will be set to a value indicating the cause of the condition (see "I-O Status").

10. The WRITE statement is used to add data records to a disk-storage file.

11. If a file is opened OUTPUT and its access mode is either sequential or dynamic/extended, a WRITE statement may be used only for creating a new file ("load mode"). At this point the data records must be transferred to the input/output system in ascending RECORD KEY order. Before execution of the WRITE statement, the RECORD KEY data item must have been set to the required value by the program.

12. If a file is opened I-O (OPEN statement with I-O option) and its access mode is either sequential or dynamic/extended, a WRITE statement is used for adding a data record to an existing file. At this point, the records may be transferred in any RECORD KEY order to the input/output system. Before the WRITE statement is executed, the program must have set the RECORD KEY item to the required value.

13. The INVALID KEY condition is triggered by the causes listed in Table 5-7.

14. Occurrence of an INVALID KEY condition indicates unsuccessful execution of the WRITE statement; the contents of the data record area are still available, and any existing FILE STATUS data item of that file is set to a value which indicates the cause of the INVALID KEY condition. Program continuation is in accordance with the rules of the INVALID KEY condition.

Programming considerations

1. The file whose data record is referred to by the WRITE statement must have been opened as OUTPUT or I-O.

2. When executing a WRITE statement, the position of the data record to be output within the file is located by means of the contents of the RECORD KEY data item.

3. Before the WRITE statement is executed, the contents of the associated key field must be set accordingly (see "RECORD KEY Clause".

Organization	Access	OPEN mode and action taken	Cause of invalid key condition
INDEXED	SEQUENTIAL or DYNAMIC/ EXTENDED	OUTPUT A record is added to a file to be newly created. "load mode").	The value of the RECORD KEY data item is less than that of the last output data record (the record keys must be sorted in ascending alphanumeric order). No more space is available on the file for writing the data record.
	RANDOM or DYNAMIC/ EXTENDED	I-O A record is added to an existing file	The contents of the RECORD KEY data item identifies a data record already existing on the file. No more space is available on the file for writing the data record.

Table 5-7 WRITE statement. Causes of INVALID KEY conditions

6 TABLE HANDLING

6.1 GENERAL DESCRIPTION

A table is a series of data items of equal length. These items are the table elements, or table items. They all have an identical structure and are stored contiguously. The entire table itself also forms a data item in COBOL terms.

Problems arising during the processing of large amounts of identically structured data can often be solved more satisfactorily by putting this data into tabular form. This allows an effective interpretation and meaningful representation of the information involved.

The identical structure of the individual table items makes clear their relationship to each other.

The individual table item occupies an easily determined physical location relative to the base of the table, i.e. to the start of the table in working storage. Thus every item can be referenced relative to the beginning of the table and does not need to be addressed under a name of its own. A table item is accessed with the aid of a table item number, or occurrence number (see "Subscripting", "Indexing").

In addition, it is possible to determine the associated occurrence number for any given value of a table item (see "SEARCH Statement").

The number of table items in a table can be changed at object time (see "OCCURS Clause with DEPENDING ON Phrase").

6

6.2 TABLE DEFINITION

A table item is indicated in the data description entry by specifying the OCCURS clause. This clause defines how may items the table contains. The name and description of the table item apply to each recurrence thereof. In the case of multi-dimensional tables, each dimension in the hierarchical structure must be given an OCCURS clause.

Example 6-1

```
01  TABLE.
    02 TABLE-ITEM PIC XXX OCCURS 20 TIMES.
```

The data item TABLE comprises 20 data items of identical length. These fields are given the name TABLE-ITEM

```
TABLE:  1. TABLE-ITEM (1)    PIC XXX.
        2. TABLE-ITEM (2)    PIC XXX.
           .
           .
           .
       20. TABLE-ITEM (20)   PIC XXX.
```

Up to three dimensions are allowed for a single table.

6.2.1 One-Dimensional Tables

The OCCURS clause is entered in the data description entry of the table item.

Example 6-2

```
01  TABLE.
    02 TABLE-ITEM OCCURS 2 TIMES.
       03 ELEMENT-ITEM-1      PIC X(4).
       03 ELEMENT-ITEM-2      PIC X(4).
```

TABLE is the name of the table.

TABLE-ITEM is the item which occurs twice within the one-dimensional TABLE.

ELEMENT-ITEM-1 and ELEMENT-ITEM-2 are items which are subordinate to TABLE-ITEM.

6.2.2 Multi-Dimensional Tables

When a data item is subordinate to a table-item within a two-dimensional table and contains an OCCURS clause, then this data item is an item within a three-dimensional table.

Example 6-3

```
01  TABLE.
    02 BLK OCCURS 2 TIMES.
       03 RECORD  OCCURS 2 TIMES.
          04 ITEM  OCCURS 2 TIMES PIC X(10).
```

BLK is an item which occurs twice within a one-dimensional table.

RECORD is an item in a two-dimensional table. It occurs twice within each occurrence of BLK.

ITEM is an item in a three-dimensional table. It occurs twice within each occurrence of RECORD.

TABLE	BLK (1)	RECORD (1, 1)	ITEM (1, 1, 1)
			ITEM (1, 1, 2)
		RECORD (1, 2)	ITEM (1, 2, 1)
			ITEM (1, 2, 2)
	BLK (2)	RECORD (2, 1)	ITEM (2, 1, 1)
			ITEM (2, 1, 2)
		RECORD (2, 2)	ITEM (2, 2, 1)
			ITEM (2, 2, 2)

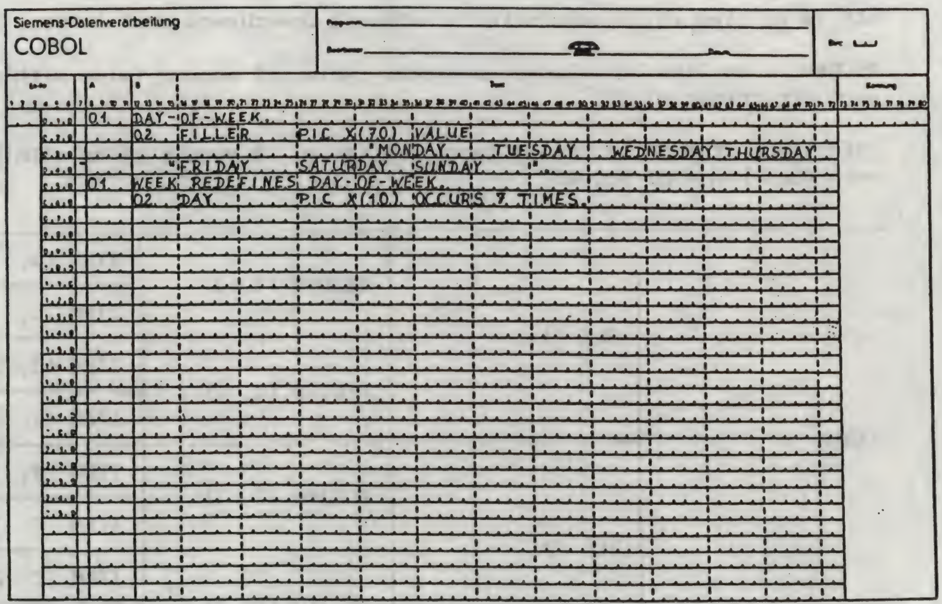
Fig. 6-1 Schematic representation of TABLE

6.2.3 Initial Values of Table Items

A VALUE clause must not appear in a record description entry with an OCCURS clause, or in any record description entry subordinate to that entry. However, here too the VALUE clause is allowed (and required) for the definition of condition-names.

Nevertheless, initial values may be assigned to a table in the WORKING-STORAGE SECTION by using the REDEFINES clause.

Example 6-4



Siemens-Datenverarbeitung
COBOL

```
       01  DAY-OF-WEEK.
           02  FILLER      PIC X(70) VALUE
                               "MONDAY    TUESDAY   WEDNESDAY THURSDAY
      -        "FRIDAY    SATURDAY  SUNDAY    ".
       01  WEEK REDEFINES DAY-OF-WEEK.
           02  DAY         PIC X(10) OCCURS 7 TIMES.
```

6.2.4 References to Table Items

All the items within a table have the same data-name. To identify table items, occurrence numbers (indexes) are appended to the data-name, enclosed in parentheses.

Example 6-5

```
    01  TABLE.
        02  ITEM   OCCURS 10 TIMES.
              .
              .
              .
           MOVE ITEM OF TABLE (8) TO ...
```

Here the eighth table item is accessed.

An occurrence number must be supplied for each dimension.

There are two techniques for referencing table items:

a) subscripting

b) indexing.

6.3 SUBSCRIPTING

One method of specifying occurrence numbers is to append one or more subscripts to the data-name. A subscript is an integer whose value represents the occurrence number of a table element or one of the items subordinate to that table element. The subscript may be represented either by an integer literal or by a data-name elsewhere defined as a numeric elementary data item without any character positions to the right of the assumed decimal point. In either case, the subscript must be enclosed in parentheses and must be followed by as many subscripts as the associated table has dimensions. A subscript must therefore be supplied for each OCCURS clause, including the OCCURS clause which contains the data-name within the defined hierarchy.

In Example 3 (three-dimensional table), the following are required:

a) a subscript for references to BLOCK.

b) two subscripts for references to RECORD.

c) three subscripts for references to ITEM.

Subscripts are written proceeding from the outermost to the innermost table. Thus, for example,

ITEM (1, 2, 2)

identifies the second ITEM within the second RECORD within the first BLOCK.

A reference to a data item must not be subscripted unless the data item is a table item, or unless it is an item or condition-name within a table item.

There are two forms of subscripting:

a) direct subscripting.

b) relative subscripting.

6.3.1 Direct Subscripting

With direct subscripting, the subscript is specified either by an integer literal or by a data-name. The data-name must be defined as a numeric elementary item with no character positions to the right of the assumed decimal point. In the preceding example, direct subscripting was used.

6.3.2 Relative Subscripting

If the name of the table item is followed by a subscript in the form:

(data-name + integer-1),

then the occurrence number required to complete the reference is calculated from the value of data-name at object time, plus integer-1.

If it takes the form:

(data-name - integer-2)

the occurrence number is obtained by subtracting integer-2 from data-name.

Relative subscripting is treated in the same manner as relative indexing. For further details see "indexing".

Subscripting and indexing must not be used together within a single reference.

6.3.3 Formats

Format 1

describes subscripting without qualification.

$$\left\{\begin{array}{l}\text{data-name}\\ \text{condition-name}\end{array}\right\}\ \text{(subscript-1 [subscript-2 [subscript-3]])}$$

Format 2

Describes subscripting with qualification.

$$\left\{\begin{array}{l}\text{data-name}\\ \text{condition-name}\end{array}\right\}\ \left\{\begin{array}{l}\underline{\text{OF}}\\ \underline{\text{IN}}\end{array}\right\}\ \text{data-name-1}\ \left[\left\{\begin{array}{l}\underline{\text{OF}}\\ \underline{\text{IN}}\end{array}\right\}\ \text{data-name-2}\right]\ \ldots$$

$$\text{(subscript-1 [subscript-2 [subscript-3]])}$$

For the explanation of and rules for qualification see "Qualification".

Rules for both formats

1. data-name (rather than data-name-1) is the name of the table-item. Its data description entry must either contain an OCCURS clause, or it must be subordinate to a data item which contains an OCCURS clause.

2. subscript-1, subscript-2, subscript-3 may be represented by

   a) an integer literal

   b) a data-name with a positive integer as its value

   c) the special TALLY register

   d) relative subscripting.

   The data-name itself may be qualified but not indexed.

3. One subscript must be specified for each OCCURS clause which is subordinate to data-name. Since a table may have one, two or three dimensions, references to an element in a table may require up to three subscripts.

4. The subscript is enclosed in parentheses. The left parenthesis immediately follows the space after the name of the table item data-name). When more than one subscript appears within a set of parentheses, these subscripts may be separated either by commas followed by at least one space, or by spaces only.

5. The subscript, or set of subscripts, identifies the table element which is to be referenced. A data-name to which one or more subscripts have been added is called a subscripted data-name or identifier.

6. When more than one subscript is used, they are entered proceeding from the outermost to the innermost table.

Programming considerations for both formats

The **subscript** may contain a plus sign. The lowest valid subscript is 1. Consequently, neither zero nor negative numbers are permitted for subscripting. The highest allowable subscript value, in any particular case, is the maximum number of occurrences of the item, as specified in the OCCURS clause.

Example 6-6 (format 1)

PROFIT (YEAR, MONTH, DAY)

Here, a data item in a three-dimensional table named PROFIT is identified by the subscripts YEAR, MONTH and DAY.

Example 6-7 (format 2)

PROFIT OF COMPANY-A (YEAR, MONTH, DAY)

In this example, PROFIT is the item being subscripted.

6.4 INDEXING

Another technique for referencing table items is indexing. Indexing is made possible by supplying the INDEXED BY phrase in the OCCURS clause.

The index does not require its own data description entry and does not represent a data item. At object time, the value of an index is a binary value representing a displacement from the beginning of the table. The value of this binary number is calculated from the number and length of the table item as follows:

binary value of index = (occurrence number - 1) * length of table item

The value of an index may be set using the SET, SEARCH or PERFORM statement. The initial value is undefined and must be set explicitly.

There are two forms of indexing:

a) direct indexing

b) relative indexing

6.4.1 Direct Indexing

Direct indexing obtains when an index is used in the manner of a direct subscript.

<u>Example 6-8</u>

```
01  TABLE.
    02  TABLE-A OCCURS 10 TIMES INDEXED BY INDEX-A PIC XX.
          .
          .
          .
      SET INDEX-A TO 7.
          .
          .
          .
      MOVE "X7" TO TABLE-A (INDEX-A).
```

Here, the record description sets up a table of 10 two-character items.

INDEX-A is declared for TABLE-A by means of the INDEXED BY phrase.

The SET statement sets the index to a value that points to the seventh item of TABLE-A. The displacement from the start of the table, i.e. the internal binary contents of INDEX-A, is (7-1) * 2 = 12. Thus, the MOVE statement transfers X7 to the seventh table element.

6.4.2 Relative Indexing

When relative indexing is used, the index is treated as a relative subscript.

When the name of a table item is followed by an index in the form

(index + integer-1),

then the required occurrence number is calculated from the value of index-name at object time, plus integer-1.

If the form

(index - integer-2),

is used, then the new occurrence number is obtained by subtracting integer-2 from the corresponding current occurrence number.

The use of relative indexing will not change the values of the index-names in the object program.

Indexing and subscripting must not be used together within a single reference.

6

6.4.3 Formats

Format 1

Describes indexing without qualification:

```
{data-name     } (index-1 [{+} integer]
{condition-name}           {-}

                 [ index-2 [{+} integer]]
                           {-}

                 [ index-3 [{+} integer]])
                           {-}
```

Format 2

Describes indexing with qualification:

```
┌data-name      ┐ ┌OF┐                 ┌OF┐
{               } {  } data-name-1 [ {  } data-name-2] ...
└condition-name ┘ └IN┘                 └IN┘

                          ┌+┐
              (index-1 [ {  } integer]
                          └-┘

                          ┌+┐
              [index-2  [ {  } integer]]
                          └-┘

                          ┌+┐
              [index-3  [ {  } integer]])
                          └-┘
```

For a description of qualification with associated rules see "Qualification".

Rules for both formats

1. data-name is the name of a table item.

   If a data-name is used with an index-name, then the data description entry of that data-name must either itself contain an OCCURS clause with an INDEXED BY phrase, or the data-name must be subordinate to a group item containing an OCCURS clause with an INDEXED BY phrase.

   For example, the reference

   TOTAL (INDEXA, INDEXB),

   implies that TOTAL belongs to a structure with two levels of OCCURS clauses, each with an INDEXED BY phrase specified.

2. Instead of index-1, index-2, index-3, it is also possible to use a numeric literal with the value of a positive integer.

3. The index is enclosed in parentheses. The left parenthesis immediately follows the space after the name of the table item (data-name). When more than one index-name appears within a set of parentheses, these index-names may be separated either by commas followed by at least one space, or by spaces only.

4. When the + integer or - integer phrase is used, the + and - characters must be preceded and followed by spaces.

5. Index-names are written proceeding from the outermost to the innermost table.

6. The lowest valid occurrence number for index-name is 1; the highest is, in any particular case, the maximum number of occurrences of the item. This maximum number is defined in the OCCURS clause. This same rule also applies to relative indexing.

7. Referencing a table item, or an item within a table item, does not change the index-name associated with this table.

8. The use of relative indexing will not change the values of indices in the object program.

Programming considerations for both formats

1. The values of indices may be stored without conversion (SET statement) in data items defined with the USAGE IS INDEX clause. These data items are then called index-data-items (see "USAGE Clause", "SET Statement").

2. An index may be modified only by a SET, SEARCH or PERFORM statement (see the descriptions of these statements).

6.5 INDEXING AND SUBSCRIPTING COMPARED

6.5.1 Availability of Occurrence Numbers for the User

Subscripting:

The occurrence number is immediately available.

Indexing:

The occurrence number is available only if preceded by a SET statement. It is calculated as follows:

value of index divided by length of data item plus 1.

6.5.2 References to Table Items

Subscripting:

At object time, the address of the table item must be calculated from subscripts (except in the case of literal subscripts), i.e. subscripted data items cannot be referenced as quickly as can data items outside the table.

Indexing:

When indexing is used, references to a table item are faster than is the case with subscripting using identifiers, since the displacement from the start of the table is already stored in the index. This is especially true of two- or three-dimensional tables.

6.5.3 Changing the Index

Subscripting:

Changing a subscript in the form of a data-name (using MOVE,ADD etc) is faster than changing an index-name using SET, since the SET statement requires that the occurrence number must first be converted to the displacement from the start of the table. This applies when the index is not being incremented or decremented by a fixed value.

Indexing:

It is faster to change an index using PERFORM or SEARCH statements than to change a subscript using available options. This applies as well to the statements SET UP BY literal and SET DOWN BY literal.

6.6 LANGUAGE ELEMENTS

Data Division language elements:

OCCURS clause (table definition)
USAGE clause with INDEX phrase (index definition)

Procedure Division language elements:

SEARCH statement
SET statement

Comparisons involving indices or index-data-items

Rules for referencing table items: see "Subscripting" and "Indexing".

Additional table handling elements, which may also be used for data items which are not part of a table are discussed in a different section of this manual (see "PERFORM Statement", Format 4).

6.6.1 Language Elements of the Data Division

OCCURS CLAUSE

Function

The OCCURS clause is used to define tables. It specifies how many items a table is to have, i.e. how often an item is to recur. All items have an identical format. The size of the table may be variable. Furthermore, indices can be supplied.

Format 1 indicates the exact number of occurrences of a data item.

Format 2 indicates a variable number of occurrences of a specified data item, ranging between a **maximum** and a **minimum** occurrence number.
The minimum occurrence number may be omitted.

Format 1

```
OCCURS integer-2 TIMES

[{ASCENDING }
 {DESCENDING} KEY IS data-name-2 [data-name-3]...]...

[INDEXED BY index-1 [index-2]...]
```

Rules for format 1

1. integer-2 represents the exact number of occurrences.
2. integer-2 must be greater than 0 and less than 32,767.
3. data-name-2 may either be the subject of the OCCURS clause (in which case data-name-3,... must not be specified), or it must be subordinate to the group item referenced by the OCCURS clause.
4. data-name-3,... must be subordinate to the group item referenced by the OCCURS clause..
5. If data-name-2 does not match the subject of the OCCURS clause, the following rules apply:
   a) data-name-2, data-name-3,... must be subordinate to the group item referenced by the OCCURS clause.
   b) data-name-2, data-name-3,... must not be described with an OCCURS clause. Furthermore, they must not be subordinate to an entry which contains an OCCURS clause.
   c) data-name-2, data-name-3,... may be qualified with OF or IN (see " Qualification").

6. data-name-2, data-name-3,... are subject to the following additional rules:

   a) Up to 12 key fields may be specified for a given table item.

   b) The sum of the lengths of all key fields associated with a table item must not exceed 256.

   c) The key fields may have the following data formats with DISPLAY, COMPUTATIONAL or COMPUTATIONAL-3.

7. index-1, index-2,... must be unique words in the program; the indices specified by the INDEXED BY phrase are not otherwise defined in the program.

8. Up to 12 indices may be specified in the INDEXED BY phrase of an OCCURS clause.

9. The OCCURS clause must not be supplied in a data description entry if that entry:

   a) has a level number 01, 77 or 88,

   b) or if that entry describes an item whose size is variable (the size of an item is variable if the data description entry of any item subordinate to it contains an OCCURS clause with the DEPENDING phrase specified).

10. The ASCENDING/DESCENDING KEY phrase indicates whether the items in the table are to be arranged in ascending or descending order, according to the values contained in data-name-2, data-name-3, etc. The data-names must be listed in descending order of significance.

    The user is responsible for seeing that the table items are sorted properly (this order is presupposed in the SEARCH ALL statement).

11. The INDEXED BY phrase indicates that it is possible, using indexing, to reference the subject of the OCCURS clause as well as every entry subordinate to that data-name. The memory allocation and its format are automatically defined by the compiler.

12. Each index contains a binary value that represents a displacement from the beginning of the table, corresponding to an occurrence number. The value is calculated as the occurrence number minus one, multiplied by the length of the entry that is indexed by the index-name (see "Indexing").

13. Except for the OCCURS clause itself, all data description clauses associated with the item whose description contains that OCCURS clause apply to each occurrence of the item described.

14. When a computational elementary item (i.e. an item whose USAGE is COMPUTATIONAL, COMPUTATIONAL-1 or COMPUTATIONAL-2) is defined with the SYNCHRONIZED clause, the compiler adds any necessary slack bytes for each occurrence of the item (see "SYNCHRONIZED Clause").

15. A VALUE clause must not be contained in a record description entry with OCCURS clause specified, or in any record description entry subordinate to that entry. However, the VALUE clause is allowed (and required) for the definition of condition-names.

Programming considerations for format 1

1. The subject of an OCCURS clause must be indexed whenever it is referenced in a statement other than a SEARCH statement.

   If the subject is the name of a group item, then all data-names belonging to the group must be indexed whenever they are used as operands.

   An indexed data-name references one particular table item. When used in a SEARCH statement, the data-name refers to the entire table.

2. The ASCENDING/DESCENDING phrase in conjunction with the INDEXED BY phrase is used in the execution of a SEARCH ALL statement (see "SEARCH Statement").

3. Before an index-name may be used as an index, it must be initialized (using a SET, SEARCH ALL or PERFORM statement with VARYING phrase).

Format 2

```
OCCURS [integer-1 TO] integer-2 TIMES
       DEPENDING ON data-name-1

[ {ASCENDING } KEY IS data-name-2 [data-name-3]... ]...
[ {DESCENDING}                                     ]

[INDEXED BY index-1 [index-2]...]
```

Rules for format 2

1. integer-1 can be 0 or a positive integer.

2. integer-2 must be greater than 0.

3. When used together, integer-1 must be less than integer-2.

4. data-name-1 must be defined as a positive, integral numeric data item.

5. data-name-1 may be qualified with OF or IN (see "Qualification").

6. data-name-1 must not be indexed, i.e. it must not be a table item or an item within a table item.

7. If data-name-1 appears in the same data record as the table whose occurrence it controls, it must appear before the variable portion of that record. That is, the data item defined by data-name-1 must precede the data record portion described by the OCCURS clause with the DEPENDING ON phrase specified.

8. The current value of data-name-1 at the object time must not exceed that of integer-2, which specifies the maximum number of occurrences.

9. When used in the DEPENDING ON phrase of the OCCURS clause for a data area of variable length, data-name-1 must be supplied with a value before this area is used in a MOVE operation as a sending or receiving field. The same data-name-1 may be used by the sending and receiving field (see example 8).

10. data-name-2 may either be the subject of the OCCURS clause (in which case data-name-3,... must not be specified) or it must be subordinate to the group item referenced by the OCCURS clause.

11. data-name-3 must be subordinate to the group item referenced by the OCCURS clause.

12. If data-name-2 does not match the subject of the OCCURS clause the following rules apply:

    a) data-name-2, data-name-3,... must be subordinate to the group item referenced by the OCCURS clause.

    b) data-name-2, data-name-3,... must not be described with an OCCURS clause. Furthermore, they must not be subordinate to an entry which contains an OCCURS clause other than the one discussed here.

    c) data-name-2, data-name-3,... may be qualified with OF or IN (see "Qualification").

13. data-name-2, data-name-3,... are subject to the following rules:

    a) Up to 12 key fields may be specified for a given table item.

    b) The sum of the lengths of all key fields associated with a table item must not exceed 256.

    c) The key fields may have the following data formats with DISPLAY, COMPUTATIONAL or COMPUTATIONAL-3.

14. index-1, index-2,... must be unique words in the program; the indices specified by the INDEXED BY phrase are not otherwise defined in the program.

15. Up to 12 indices may be specified in the INDEXED BY phrase of an OCCURS clause.

16. The OCCURS clause must not be supplied in a data description entry if that entry:

    a) has a level number 01, 77 or 88,

    b) or if that entry describes an item whose size is variable (the size of an item is variable if the data description entry of any item subordinate to it contains an OCCURS clause with the DEPENDING phrase specified).

17. A data description entry containing an OCCURS clause with the DEPENDING ON phrase may only be followed, within that record description, by data description entries which are subordinate to it.

18. The DEPENDING ON phrase specifies that the data item described by the OCCURS clause has a variable number of occurrences. The number of occurrences is controlled at object time by the value of data-name-1.

19. integer-1 and integer-2 specify the minimum and maximum number of occurrences, respectively. The value of the data item specified by data-name-1 must range between integer-1 and integer-2.

20. If the value of data-name-1 is reduced at object time, the contents of the data items with occurrence numbers greater than the new value of data-name-1 are undefined.

21. When referencing a group item to which an entry with an OCCURS DEPENDING ON clause is subordinate, the only portion of the table item which can be used in the operation is that portion defined by the contents of data-name-1.

22. If, within a record description entry, a data area follows data items with the DEPENDING ON phrase, but is not subordinate to those items, then its position depends on the current values of data-name-1 in the preceding DEPENDING ON phrases.

    If the value of data-name-1 is changed (i.e. change of table length), the position of these data areas are shifted accordingly. However, their original contents are **not** shifted (see Example 10).

23. The ASCENDING/DESCENDING KEY phrase indicates whether the items in the table are to be arranged in ascending or descending order, according to the values contained in data-name-2, data-name-3, etc. The data-names must be listed in descending order of significance.

    The user is responsible for seeing that the table items are sorted properly (this order is presupposed in the SEARCH ALL statement).

24. The INDEXED BY phrase indicates that it is possible, using indexing, to reference the subject of the OCCURS clause as well as every entry subordinate to that data-name. The memory allocation and its format are automatically defined by the compiler.

25. Each index contains a binary value that represents a displacement from the beginning of the table, corresponding to an occurrence number. The value is calculated as the occurrence number minus one, multiplied by the length of the entry that is indexed by the index-name (see "Indexing").

26. Except for the OCCURS clause itself, all data description clauses associated with the item whose description contains that OCCURS clause apply to each occurrence of the item described.

27. When a computational elementary item (i.e. an item whose USAGE is COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2) is defined with the SYNCHRONIZED clause, the compiler adds any necessary slack bytes for each occurrence of the item (see "SYNCHRONIZED Clause").

28. Any entry that contains an OCCURS clause with the DEPENDING ON phrase specified, or has a subordinate entry with an OCCURS... DEPENDING ON clause, may not contain a REDEFINES clause unless it is an 01-level entry in the WORKING-STORAGE section.

29. When a record description entry contains the OCCURS clause with DEPENDING ON phrase specified, then the VALUE clause must not be specified in any data description entries subordinate to that entry, except in the case of condition-names.

30. Records are variable-length when format 2 is specified in a record description entry and the associated file description entry contains the RECORD clause with VARYING phrase.

    If DEPENDING ON is omitted, the contents of the data-name-1 specified in the OCCURS clause must be set to the number of recurrences prior to execution of each RELEASE, REWRITE or WRITE statement.

Programming considerations for format 2

1. The subject of an OCCURS clause must be indexed whenever it is referenced in a statement other than a SEARCH statement.

   If the subject is the name of a group item, then all data-names belonging to the group must be indexed whenever they are used as operands.

   An indexed data-name references one particular table item. When used in a SEARCH statement, the data-name refers to the entire table.

2. The ASCENDING/DESCENDING phrase in conjunction with the INDEXED BY phrase is used in the execution of a SEARCH ALL statement (see "SEARCH Statement").

3. Before an index-name may be used as an index, it must be initialized (using a SET, SEARCH ALL or PERFORM statement with VARYING phrase).

6

Example 6-9

Supplying a value to data-name-1 in OCCURS DEPENDING ON, using a MOVE operation.

Given the following data definition:

```
WORKING-STORAGE SECTION.
01  ITEM-A.
    02  COUNTER-A PIC 9.
    02  DATA-A.
        03  CHARACTER-A PIC X OCCURS 1 TO 9
            DEPENDING ON COUNTER-A

01  ITEM-B.
    02  COUNTER-B PIC 9.
    02  DATA-B.
        03  CHARACTER-B PIC X OCCURS 1 TO 9
            DEPENDING ON COUNTER-B.
```

The following MOVE operations are to be performed:

Case a: Sending field larger than receiving field

```
MOVE 6 TO COUNTER-A,
MOVE 3 TO COUNTER-B,
MOVE ITEM-A TO ITEM-B.
```

Contents following MOVE operation:

```
ITEM-A: 6ABCDEFGHI
ITEM-B: 6ABCMNOPQR
```

Case b: Sending field smaller than receiving field (Contents of both fields as they were before Case a)

```
MOVE 3 TO COUNTER-A,
MOVE 6 TO COUNTER-B,
MOVE ITEM-A TO ITEM-B.
```

Contents following MOVE operation:

```
ITEM-A: 3ABCDEFGHI
ITEM-B: 3ABC   PQR
```

The MOVE operations proceed according to the rules for alphanumeric moves (see "MOVE Statement").

Example 6-10

OCCURS DEPENDING ON data-name-1

Given the following data definition:

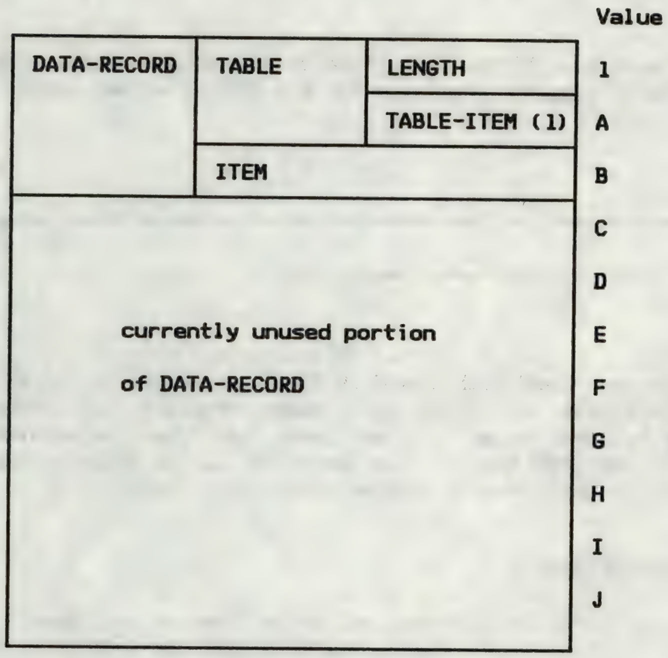
```
     WORKING-STORAGE SECTION.
     01  DATA-RECORD.
         02  TABLE.
             03  LENGTH      PIC 9.
             03  TABLE-ITEM  PIC X OCCURS 1 TO 9
                             DEPENDING ON LENGTH.

         02  ITEM      PIC X.
```

If the current value of LENGTH is 9, the following starting position of the items results:

			Value
DATA-RECORD	TABLE	LENGTH	9
		TABLE-ITEM (1)	A
			B
			C
			D
			E
			F
			G
			H
		TABLE-ITEM (9)	I
	ITEM		J

After MOVE 1 TO LENGTH

the length of the table and hence the position of ITEM is changed:



The data item LENGTH now has the value 1; the data item ITEM has the value B.

USAGE CLAUSE

Function

The USAGE clause with the INDEX phrase specified indicates that a data item is to be used as temporary storage for the values of an index.

Format

[USAGE IS] INDEX

Rules

An elementary item described with USAGE IS INDEX is called an index data item. This is a data item (not necessarily associated with any table) which may be used to save values of index-names for future reference. An index data item is assigned the value of an index by the SET statement. The value of an index data item is not an occurrence number.

Programming considerations

1. The USAGE clause with INDEX phrase may be written at any level. If a group item is described with USAGE IS INDEX, the elementary items in the group are all index data items; the group itself is not an index data item.

2. An index data item can be referenced directly only in a SEARCH statement, in a SET statement, in a relation condition, in the USING phrase of the Procedure Division header, or in the USING phrase of a CALL statement.

3. An index data item cannot be a conditional variable.

4. An index data item may be part of a group which is referenced in a MOVE statement or an input/output statement. When such statements are executed, however, the contents of the index data item is not converted.

5. SYNCHRONIZED, JUSTIFIED, PICTURE, BLANK WHEN ZERO or VALUE clauses cannot be used to described group items or elementary items described with USAGE IS INDEX.
   However, the compiler allows the SYNCHRONIZED clause to be used with the USAGE IS INDEX clause.

Example 6-11

```
      02  ALPHA PICTURE X(9) OCCURS 5 INDEXED BY A-NAME.
      .
      .
      .
  77  A-INDEX USAGE IS INDEX.
      .
      .
      .
      SET A-NAME TO 3.
      .
      .
      .
      SET A-INDEX TO A-NAME.
```

Here the index data item A-INDEX is set to the current value of the index-name A-NAME, i.e. the occurrence number (3) minus 1, multiplied by the length of the entry (9) = 18.

6.6.2 Language Elements of the Procedure Division

SEARCH STATEMENT

Function

The SEARCH statement is used to search a table for a table item that satisfies a specified condition, and to set the value of the associated index to the corresponding occurrence number.

Format 1 is used to perform a serial search of a table. The search for identifier-1 begins at the current value of the index assigned to identifier-1.

Format 2 is used to perform a binary search of table.

Format 1

```
                              ┌        ┌index-1       ┐┐
SEARCH identifier-1   │VARYING <                        >│
                              └        └identifier-2 ┘┘

         [AT END imperative-statement-1]

          WHEN condition-1 ┌imperative-statement-2┐
                           <                        >
                           └NEXT SENTENCE          ┘

        ┌WHEN condition-2 ┌imperative statement-3┐┐
        │                 <                        >│...
        └                 └NEXT SENTENCE          ┘┘
```

Rules for format 1

1. identifier-1 specifies the table to be searched.

2. The data description entry of identifier-1 must include an OCCURS clause with an INDEXED BY phrase specified.

3. identifier-1 must not be subscripted or indexed.

4. index-1 or identifier-2 specifies an item whose value is to be varied during execution of the SEARCH statement.

5. index-1 may be one of the indices for identifier-1, or an index for another table entry.

6. identifier-2 must therefore be described as an index data item (USAGE IS INDEX), or it must be a fixed-point numeric item described as an integer.

7. The AT END phrase specifies a statement which is to be executed if the search is unsuccessful.

8. Each condition, condition-1, condition-2, etc. must be a valid condition (see "Conditions").

9. condition-1, condition-2,... specify the conditions to be satisfied during the execution of the SEARCH statement.

10. imperative-statement-2 (or -3) or NEXT SENTENCE specifies an action to be taken when the associated WHEN condition is satisfied: control passes to the imperative-statement or to the next sentence (that is, the statement following the SEARCH statement), depending on the option specified.

11. A serial search of a table begins at the table item pointed to by the index associated with identifier-1 for search purposes.

12. If, at the start of a SEARCH statement, the value of the index associated with identifier-1 is greater than the highest permissible occurrence number for identifier-1, the search will terminate immediately. If the AT END phrase is specified, imperative-statement-1 is executed. If this phrase is omitted, processing continues with the next statement.

13. If, at the start of a SEARCH statement, the value of the index associated with identifier-1 corresponds to a valid occurrence number for identifier-1, the serial search takes place as follows:

    a) The WHEN conditions are evaluated in the order in which they are written.
    b) If none of the conditions is satisfied, the index-name for identifier-1 is incremented to refer to the next occurrence of a table element; and step a) is repeated, unless the new value of the index corresponds to an occurrence number outside the valid range, in which case step d) is performed.
    c) If one of the WHEN conditions is satisfied, the search terminates immediately. The index points to the table element that satisfied the condition. The imperative statement associated with that condition is executed.
    d) If the end of the table is reached without the WHEN condition being satisfied, the search terminates. If the AT END phrase is specified, imperative-statement-1 is executed. If this phrase is omitted, control passes to the next sentence.

14. When identifier-1 is a data item subordinate to a data item that contains an OCCURS clause, then two- or three-dimensional tables can be searched. In this case, an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Execution of a SEARCH statement modifies only the setting of the index associated with identifier-1 (and, if present, of index-1 or identifier-2). Therefore, in order to search a two- or three-dimensional table, a SEARCH statement must be executed for each possible value of the superordinate index.

    Before the SEARCH statements are executed, the corresponding indices must be preset with the required values by means of SET statements or with PERFORM VARYING (see example 13).

15. If, in the AT END phrase and the WHEN conditions, none of the imperative-statements specified terminates with a GO TO statement, then control will pass to the next statement after the imperative statement is executed.

16. If the VARYING index-1 phrase is specified, the following takes place:

    a) If index-1 is one of the indices for identifier-1, index-1 is used for the search. No other indices are incremented.
    b) If index-1 is an index for another table entry, the first, or only, index associated with identifier-1 is used for the search. When the index associated with identifier-1 is incremented, index-1 is simultaneously incremented to represent the next item in its table.

17. If the VARYING identifier-2 phrase is specified, the following actions take place:

    a) The first, or only, index associated with identifier-1 is used for the search.
    b) When the index associated with identifier-1 is incremented, identifier-2 is simultaneously incremented.
    c) If identifier-2 is a numeric item, it is incremented by 1.
    d) If identifier-2 is an index data item, it is incremented by a value equal to that used to increment the index associated with identifier-1.

    If the VARYING phrase is not specified, the first, or only, index associated with identifier-1 (i.e. defined in the INDEXED BY phrase of the data description entry of identifier-1) will be used for the search.

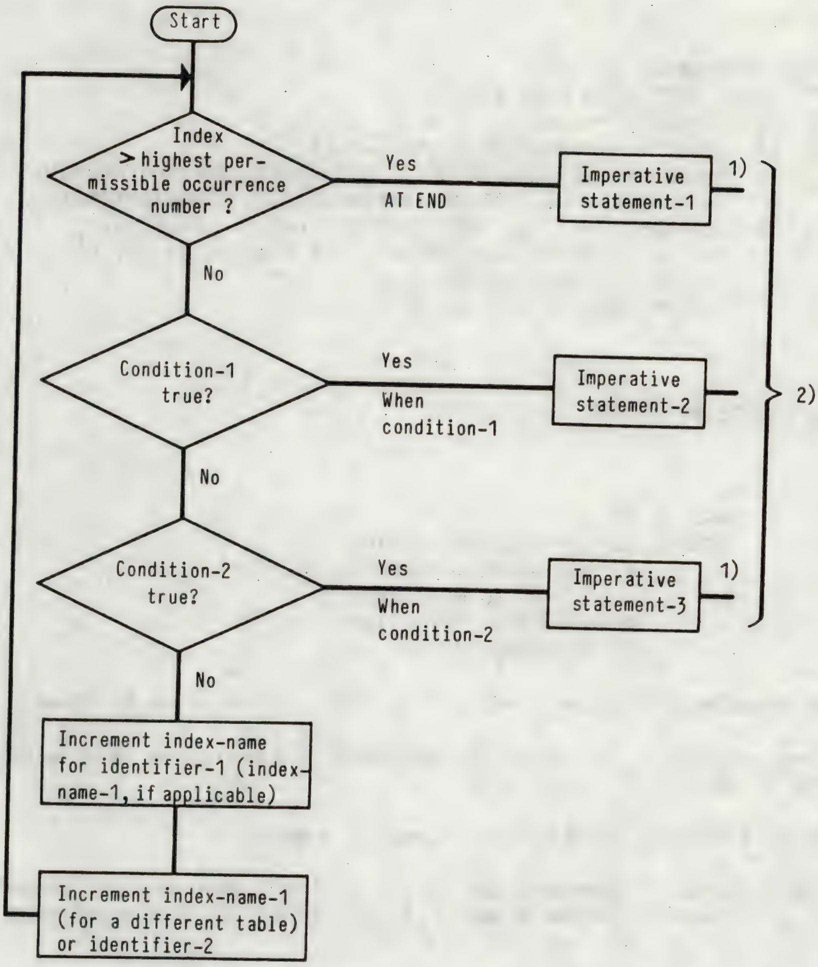


Fig. 6-2 Format-1 SEARCH statement containing two WHEN conditions

1) These operations are included only when specified in the SEARCH statement.

2) Each of these control transfers is to the next sentence, unless the imperative-statement ends with a GO TO statement.

Example 6-12

```
     WORKING-STORAGE SECTION.
     77  INPUT-LETTER PICTURE A.
     01  LETTER-WEIGHT-TABLE.
         02  ORIGINAL-DEFINITION PICTURE X(78)
             VALUE IS "A01B03C03D02E01F04G02H04I01J08K05L01M03N010
-                        "01P03Q10R01S01T01U01V04W04X08Y04Z10".
         02  SECOND-DEFINITION REDEFINES ORIGINAL-DEFINITION.
             03  LETTER-TABLE OCCURS 26 TIMES INDEXED BY LI, PI.
                 04  LETTER PICTURE A.
                 04  POINTS PICTURE 99.
                     .
                     .
                     .
     PROCEDURE DIVISION.
                     .
                     .
                     .
             SET PI TO 1.
             SEARCH LETTER "ILLEGAL LETTER",
                    AT END DISPLAY "ILLEGAL LETTER",
                           GO TO NEXT LETTER
                    WHEN LETTER (PI) = INPUT-LETTER
                    GO TO FOUND.
```

In this example, the table named LETTER-TABLE consists of 26 items.

Each item contains a letter of the alphabet followed by a value associated with the letter.

The table is indexed by the index-names LI and PI.

The SEARCH statement searches the table for the item whose LETTER matches the current contents of the area called INPUT-LETTER. The associated index is PI.

The search starts at the beginning of the table since PI points to the first table item.

If the search is successful, the statement GO TO FOUND is executed. In this case, the index PI points to the item satisfying the condition. For example, if INPUT-LETTER contains B, the index points to the second table item.

If the search is unsuccessful, the statements in the AT END clause are executed.

Example 6-13

```
IDENTIFICATION DIVISION.
PROGRAM-ID.            SEARCH1.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  MATRIX.
    02  LINE OCCURS 10 INDEXED I.
        03  ITEM PIC X OCCURS 10 INDEXED J.
01  LOCATED PIC X(4).
PROCEDURE DIVISION.
START.
    DISPLAY "ENTER ENTIRE MATRIX LINE BY LINE" UPON TERMINAL.
    ACCEPT MATRIX FROM TERMINAL.
    IF   MATRIX    = "END"
    THEN STOP RUN.
    MOVE "NO" TO LOCATED.
    PERFORM SEARCH-IN-LINE VARYING I FROM 1 BY 1
         UNTIL I > 10 OR LOCATED = "YES".
    IF   LOCATED = "YES"
    THEN DISPLAY "LOCATED" UPON TERMINAL
    ELSE DISPLAY "NOTHING THERE" UPON TERMINAL.
    GO TO START.
SEARCH-IN-LINE.
    SET J TO 1.
    SEARCH ITEM VARYING J
       AT END MOVE "NO" TO LOCATED
       WHEN ITEM (I, J) = "G"
            MOVE "YES" TO LOCATED.
```

In the SEARCH-IN-LINE, the I-th line of the MATRIX is searched.

For search purposes, index-name J is varied from 1 to 10; hence items ITEM(I,1),..., ITEM(I,10) are compared with "G".

This search operation is performed for all 10 lines of the matrix (PERFORM) until a "G" is encountered.

Thus, the entire matrix is searched for "G".

If a "G" is encountered, I points to the first line, and J to the first column, in which a "G" appears.

Format 2

```
SEARCH ALL identifier-1
           [AT END imperative-statement-1]

                          ┌imperative-statement-2┐
           WHEN condition {                      }
                          └NEXT SENTENCE         ┘
```

Rules for format 2

1. identifier-1 specifies the table to be searched.

2. The description of identifier-1 must contain an OCCURS clause with the INDEXED BY and ASCENDING/DESCENDING phrases specified.

3. identifier-1 must not be subscripted or indexed.

4. The AT END phrase specifies the statement to be executed if condition cannot be satisfied for any setting of the index within the valid range (see rule 10).

5. condition specifies the condition that must be satisfied during the execution of the SEARCH statement.

6. condition must be one of the following types of conditions (see also under "Conditions"):

   a) A relation condition incorporating the EQUAL TO or the equal sign (=) relation.

      Either the subject or the object (but not both) of the relation condition must consist solely of one of the data-names that appear in the ASCENDING/DESCENDING phrase of identifier-1. Each data-name must be indexed by the first index associated with identifier-1.

   b) A condition-name condition in which the VALUE clause describing the condition-name contains only a single literal.

      The conditional variable associated with the condition-name must be one of the data-names that appear in the ASCENDING/DESCENDING clause of identifier-1.

   c) A compound condition formed from simple conditions of the types described above, with AND as the only connective.

7. Any data-name that appears in the ASCENDING/DESCENDING clause of identifier-1 may be tested in condition. However, all data-names in the ASCENDING/DESCENDING clause preceding the data-name to be tested, must also be tested in condition. No other tests need be made in condition.

8. imperative-statement-2 or NEXT SENTENCE specifies an action to be taken when condition is satisfied. Control passes to imperative-statement-2 or to the statement following the SEARCH statement.

9. The **first** index associated with identifier-1 is used for the search. This index does not have to be initialized with a SET statement since its initial value is ignored for the search.

10. The SEARCH ALL statement is executed as follows, whereby the table must be arranged in ascending or descending order of the key fields listed in the ASCENDING/DESCENDING option:

    a) During the search the value of the index associated with identifier-1 is varied.

    b) This setting is never less than the value corresponding to the first table item, and never greater than the value corresponding to the last table item.

    c) If condition cannot be satisfied for any setting of the index within this permitted range, control is passed to imperative-statement-1 if the AT END phrase is specified, or to the next statement if the AT END phrase is omitted.

11. If condition can be satisfied, the index points to the table item satisfying the condition. Control then passes to imperative-statement-2 or to the next sentence.

12. When identifier-1 is a data item subordinate to a data item that contains an OCCURS clause, then two- or three-dimensional tables can be searched. In this case, an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Execution of the SEARCH ALL statement modifies only the setting of the index-name associated with identifier-1. Therefore, in order to search a two- or three-dimensional table, a SEARCH ALL statement must be executed for each possible value of the superordinate index.

13. If, in the AT END phrase and the WHEN conditions, none of the statements specified terminates with a GO TO statement, then control will pass to the next statement after the imperative-statement is executed.

Example 6-14 (WHEN conditions)

Data Division entries:

```
        .
        .
        .
        .

            .
            .
            .
    77  A-VALUE PICTURE 9.
            .
            .
            .
        02  TABLE-ITEM OCCURS 5 TIMES ASCENDING KEY IS A,B,C;
                    INDEXED BY I.
            03  A  PICTURE 99.
            03  B  PICTURE 9.
                88 UNDER-30 VALUE 1.
                88 OVER-30 VALUE 2.
            03  C  PICTURE 9.
            .
            .
```

Valid WHEN phrases (in Procedure Division)

```
WHEN A(I) = 10
WHEN A(I) = 20 AND UNDER-30(I)
WHEN A(I) = 15 AND OVER-30(I) AND C(I) = A-VALUE
```

Example 6-15

```
WORKING-STORAGE SECTION.
77  IPD PIC 9(3).
77  INPUT-LN PIC 9(6).
01  EMPLOYEE-TABLE.
    02  PERSON OCCURS 2000 TIMES INDEXED BY PI,
                 ASCENDING KEY IS DEPARTMENT LIFE-NUMBER.
        03  DEPARTMENT  PIC 9(3).
        03  LIFE-NUMBER PIC 9(6),
        03  NAME        PIC X(20).
PROCEDURE DIVISION.
    ...

        SEARCH ALL PERSON
              AT END GO TO MISSING-PERSON
              WHEN DEPARTMENT (PI) = IPD
                AND LIFE-NUMBER  (PI) =
                INPUT-LN GO TO FOUND.
```

In this example, the table named PERSON consists of 2000 elements.

Each element in the table consists of a 3-byte numeric item called DEPARTMENT, a 6-byte numeric item called LIFE-NUMBER and a 20-byte alphanumeric item called NAME.

The table is arranged in ascending order by DEPARTMENT and, within DEPARTMENT, in ascending order by LIFE-NUMBER.

The first portion of the table might have the following contents:

DEPARTMENT	LIFE-NUMBER	NAME
101	123456	ADAM, D.
101	234561	LANGEWIESCHE, W.
101	523618	EBERLE, F.
183	200305	DAUTZENBERG, K.
183	328512	REINHARDT, M.
183	433333	GRUEN, L.
183	987245	RICHTER, L.
557	328835	SCHMIDT, S.
557	775247	ALBRECHT, N.

The SEARCH statement searches the table for an element whose DEPARTMENT matches the current contents of the area called IPD, and whose LIFE-NUMBER matches the current contents of the area called INPUT-LN. If the search is successful, the statement GO TO FOUND is executed. The index-name, PI, points to the element satisfying the condition.

For example, if IPD contains 183 and INPUT-LN contains 328512, the index-name points to the fifth element of the table.

If the search is unsuccessful, the statement GO TO MISSING-PERSON is executed.

SET STATEMENT

Function

The SET statement defines reference points for table handling operations by setting index-names which point to table items. The SET statement must be used for initializing an index-name prior to the execution of a SEARCH statement. The SET statement can be used to change the status of external switches or to set the value of conditional variables.

Format 1 sets an integer data item, index-name or index data item to a specified value.

Format 2 increments or decrements the value of an index-name to represent a new occurrence number.

Format 3, Format 4 change the status of external switches or set the value of conditional variables (see "SET Statement" in chapter 2).

Format 1

```
    ⎧index-1             [index-2]...   ⎫     ⎧index-3       ⎫
SET ⎨                                    ⎬ TO  ⎨identifier-3  ⎬
    ⎩identifier-1[identifier-2]...      ⎭     ⎩literal-1     ⎭
```

Rules for format 1

In the following notes, all references to index-1 and identifier-1 apply equally to index-2 and identifier-2, respectively.

1. Each index must be specified in an INDEXED BY phrase of an OCCURS clause.

2. Each identifier must name either an index data item or a fixed-point numeric elementary item described as an integer.

3. Any literal must have a value ≤ 32767.

4. literal-1 must be a non-negative integer and may contain a plus sign.

   The compiler also permits 0 and ZERO as values for literal-1.

   Since 0 is not a valid occurrence number, index-1 must in this case be set to a permitted value (e.g. via SET UP BY) before it can be used for indexing.

5. Indices or identifiers preceding the TO specify the item whose value is to be set.

6. index-3, identifier-3 and literal-1 (which follow the TO) specify the value to which the receiving field (e.g. index-1) is to be set.

7. When executing the SET statement, one of the following actions occurs:

   a) index-1 is set to refer to the table item that corresponds, in occurrence number, to the table item referenced by index-3, identifier-3, or literal-1.

      If identifier-3 names an index data item, or if index-3 is related to the same table as index-1, no conversion takes place.

   b) If identifier-1 is an index data item, it is set equal to either the contents of index-3 or identifier-3, where identifier-3 is also an index data item. No conversion occurs. Literal-1 cannot be used in this case.

   c) If identifier-1 is not an index data item, it is set to an occurrence number that corresponds to the value of index-3. Neither identifier-3 nor literal-1 can be used in this case.

This process is repeated for index-2, identifier-2, etc., if specified. Each time, the value of index-3 or identifier-3 is used as it was at the beginning of the execution statement. Any subscripting or indexing associated with identifier-1 is evaluated before the value of the respective data item is changed.

The table below indicates the validity of various operand combinations in the SET statement. The letters following the slashes refer to the rules listed above under point 7; for example, valid/c indicates that a combination of items is valid according to rule c) above.

Sending field	Receiving field		
	Integer data item PIC9(n)	Index-name INDEXED BY	Index-name USAGE IS INDEX
Integer literal	no/c	valid/a	no/b
Integer data item	no/c	valid/a	no/b
index-name	valid/c	valid/a	valid/b *
Index data item	no/c	valid/a *	valid/b *

* = no conversion takes place

Table 6-1 Valid uses of the SET statement

8. If index-3 is used, its value prior to execution of the SET statement must correspond to an occurrence number of the element in the associated table.

Example 6-16

```
Data Division Entries:
   02 TABLE-A PICTURE XXX OCCURS 50,
              INDEXED BY IN-A1, IN-A2, IN-A3,
   02 TABLE-B PICTURE XX OCCURS 55,
              INDEXED BY IN-B.
77 ID-1 USAGE IS INDEX.
77 D-1 PICTURE IS S999, USAGE IS COMPUTATIONAL-3.
```

Procedure Division Statements:

See the tabular presentation below:

Statement	Operands used	Action taken 1)
SET IN-A2 TO IN-A1	Index-name set to index-name	Simple move (displacement to displacement).
SET IN-A1 TO D-1	Index-name set to numeric data item	Multiply (numeric item minus 1) by item length to get displacement.
SET IN-A1 TO ID-1	Index-name set to index data item	Simple move (displacement to displacement).
SET IN-A1 TO 4	Index-name set to literal	Multiply (literal minus 1) 2) by item length to get displacement.
SET ID-1 TO IN-A1	Index data item set to index-name	Simple move (displacement to displacement)
SET D-1 TO IN-A1	Numeric data item set to index-name	Divide displacement by item length and add 1, to get occurrence number.
SET IN-B TO IN-A1	Index-name set to index-name (different tables)	Divide displacement (i.e., contents of IN-A1) by item length for TABLE-A and add 1, to get occurrence number. Then, multiply (occurrence number minus 1) by item length for TABLE-B, to get displacement.

1) See "Indexing" for the relationships between occurrence number and displacement.

2) Calculated at compilation time; simply move during program run.

Example 6-17

```
02  TABLE-A OCCURS 50 TIMES
              INDEXED BY IND-1,IND-2,PIC 999.
    ...
SET IND-1 TO 5.
SET IND-2 TO 7.
SET IND-1,TABLE-A (IND-1) TO IND-2.
```

The third SET statement is equivalent to the following two statements, which are executed in the order in which they appear:

Statement	Action
SET IND-1 TO IND-2	IND-1 is set to the occurrence number 7, the current value of IND-2.
SET TABLE-A (IND-1) TO IND-2	Since the first SET statement sets IND-1 to 7, TABLE-A (IND-1) = TABLE-A (7). Thus, this statement sets TABLE-A (7) to 7.

Format 2

$$\underline{\text{SET}}\ \text{index-4}\ [\ \text{index-5}]\ \ldots \left\{\begin{matrix}\underline{\text{UP BY}}\\ \underline{\text{DOWN BY}}\end{matrix}\right\} \left\{\begin{matrix}\text{identifier-4}\\ \text{literal-2}\end{matrix}\right\}$$

Rules for format 2

1. Each index must be specified in an OCCURS clause with INDEXED BY phrase.

2. index-4, index-5,... indicates the storage index whose value is to be changed.

3. identifier-4 must be a fixed-point numeric elementary item described as an integer.

4. literal-2 must be an integer unequal to 0, and may contain a plus sign.

5. Any literal must have a value ≤ 32767.

6. The UP BY or DOWN BY phrase indicates that the contents of the index specified is to be either incremented (UP BY) or decremented (DOWN BY) by a value which corresponds to the occurrence number representing the value of identifier-4 or literal-2.

<u>Example 6-18</u>

```
02  TABLE-A PICTURE X(20) OCCURS 5 INDEXED BY IND-1.
    .
    .
    .
SET IND-1 TO 4.
SET IND-1 UP BY 2.
SET IND-1 DOWN BY 3.
```

The first SET statement sets index IND-1 to occurrence number 4; the second, to 6 (i.e. 4 + 2); and the third, to 3 (i.e. 6 - 3).

COMPARISONS INVOLVING INDEX-NAMES AND/OR INDEX DATA ITEMS

The allowable relation tests involving indices and/or index data items as well as the data items compared in each case are listed below (for a summary of all relation tests permitted, see "Conditions").

1. Two indices:
   The occurrence numbers corresponding to the two indices are compared.

2. One index and one numeric integer (the integer may be a numeric data item or a numeric literal):
   The numeric integer is treated as an occurrence number, and is compared with the occurrence number corresponding to the index.

3. One index data item and one index or other index data item:
   The current values of the items (i.e. the displacements from the beginning of the table) are compared.

4. The result of a comparison of an index data item with any data item or literal not specified above will be undefined.

First operand	Second operand			
	Index	Index data item	Data-name (numeric integer only)	Numeric literal (integer only)
Index	Compare occurrence number	Compare without conversion	Compare occurrence number with numeric integer of data-name	Compare occurrence number with literal
Index date item	Compare without conversion	Compare without conversion	Illegal	Illegal
Data-name (numeric integer only)	Compare occurrence number with numeric integer of data-name	Illegal	Compare numbers	Compare numbers
Numeric literal (integer only)	Compare occurrence number with literal	Illegal	Compare numbers	Compare numbers

Table 6-2 Indices and index data items - valid comparisons

7 PROGRAM COMMUNICATION

7-1 GENERAL DESCRIPTION

Program Communication statements permit communications between object programs which were compiled separately.

The following paragraphs describe the two types of programs, namely calling and called programs, as well as the use of the program communication statements.

Calling program:

The **calling program** passes control to the called program, through the CALL statement. Also, it may supply a list of data-names which are defined in the calling program and may be referred to by the called program.

Called program:

Control is transferred to the **called program** at the beginning of the Procedure Division or at an ENTRY statement. The EXIT PROGRAM statement is used to return control to the calling program, i.e. to the statement immediately following the CALL statement.

A called program may modify data items in a calling program. These items are defined in both the calling and called programs. No storage space is reserved for the items in the called program; instead, the calling program passes the addresses of its items to the called program. Thus, when the called program modifies these items, it modifies them in the calling program.

A called program may also be a calling program; that is, a called program can, in its turn, call another program. A called program may not, however, call the program from which it was called.

The sequence of returns from a series of called programs must be the reverse of the calling sequence; that is, the return from a called program will always lead back first to the calling program from which the immediate transfer of control was made, rather than to any previous calling program.

The statements of Program Communication are:

the Procedure Division header,
the CALL statement,
the EXIT PROGRAM statement,
the ENTRY statement.

7

7.2 LANGUAGE ELEMENTS

Summary

Source program division	Section or statement	Meaning
DATA DIVISION	LINKAGE SECTION	in the called program: to define data passed from the calling program. For more details see Linkage Section.
PROCEDURE DIVISION	Procedure Division Header	in the called program: to define the standard entry point into a called program; optionally to specify a list of data-names (defined in the Linkage Section) indicating that data is transferred from the calling program.
	CALL statement	in the calling program: to call a subprogram.
	ENTRY statement	in the called program: to define an entry point; optionally to specify a list of data-names that identify data transferred from the calling program.
	EXIT PROGRAM statement	in a called program: to return control to the calling program.

7.2.1 Data Division Language Elements

LINKAGE SECTION

Function

The LINKAGE SECTION is located in a called program. It describes data items in the calling program, which may also reference the called program.

Format

```
A       margin indication
|
v
LINKAGE SECTION.
{[77-level description entry.]}
{[record description entry.  ]}...
```

7.2.2 Procedure Division Language Elements

PROCEDURE DIVISION HEADER

Function

In a called program (subroutine), the Procedure Division header determines the standard entry point. Optionally, data-names may be specified if data is transferred from the calling program as a parameter.

Format

<u>Procedure</u> <u>Division</u> [<u>USING</u> data-name-1 [,data-name-2]...].

Rules

1. The USING phrase may be written only if the object program is called by a CALL statement and the CALL statement in the calling program includes a USING phrase. The number of operands in the corresponding USING phrases must be identical; otherwise, the result is unpredictable.

2. Each operand supplied in the USING phrase of the Procedure Division header must be defined in the LINKAGE section of the program containing this header, and must have a level number of either 01 or 77.

Programming considerations

1. The standard entry point within a called program is determined by the Procedure Division header (see also the ENTRY statement for nonstandard entry points). In order to transfer control from a calling program to that entry point, the calling program must contain a CALL statement. The program-name supplied in this CALL statement must be the same as the program-name specified in the PROGRAM-ID paragraph of the Identification Division of the called program.

2. The USING phrase, when specified, has the effect that data-name-1 of the Procedure Division header in the called program and identifier-1 in the USING phrase of the CALL statement in the calling program refer to the same set of data, which is equally available both to the called and the calling program. It is not necessary for the names to be identical. A similar relationship also exists between data-name-2,... in the USING phrase of the called program and identifier-2,..., in the USING phrase of the CALL statement in the calling program. A data-name may appear only once in the Procedure Division header of the called program, whereas the same identifier may occur several times in the USING phrase of the CALL statement.

3. In the called program, the operands of the USING phrase are treated according to the data description supplied in the LINKAGE section.

4. If no data-names are supplied to a called program, there is no indication in the Procedure Division header whether or not his is a called program. Since the compiler treats the two types of programs differently, it must recognize the type of program at compile time.

   The compiler considers a program to be a called program if it satisfies at least one of the following conditions:

   1. The Procedure Division header includes a USING phrase.

   2. The program includes at least an EXIT PROGRAM statement.

   If neither condition is satisfied, the program cannot run as a called program at execution time, unless it contains an ENTRY statement.

   If the above conditions are satisfied, the program may run both as a called program and as a calling program at execution time. An exception to this is the first program (called for execution by the system); it **must not** contain a USING phrase in the Procedure Division header.

CALL STATEMENT

Function

The CALL statement passes control to a called program. Optionally, operands may be specified to enable the called program to access data of the calling program.

Format

```
CALL literal [USING {identifier-1} [, {identifier-2} ...]]
                    {file-name-1 }    {file-name-2 }
```

Rules

1. literal must be a nonnumeric literal.

   literal must be a valid program-name; that is, it must begin with an alphabetic character, may contain no characters other than letters and digits, and must have a length of up to 8 characters.

2. The program-name specified by the literal must be unique in the object programs which are being linked to one executing program. Also, it must not be the same as the first 8 characters of program-name in the PROGRAM-ID paragraph of the program containing the CALL statement.

3. The USING phrase in a CALL statement may be supplied only if a USING phrase was written either in the associated Procedure Division or in the ENTRY statement of the called program. Each USING phrase must have the same number of operands, otherwise the result will be unpredictable.

4. Every identifier specified in the USING phrase must be defined in the FILE SECTION, WORKING-STORAGE SECTION, LINKAGE SECTION or SUB-SCHEMA SECTION. It may have level number 01 or level number 77. However, the compiler allows every level number except 88. In order to align elementary items with USAGE COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2 in the LINKAGE section, all 01-level items included are aligned on doubleword boundaries. Consequently, the user must ensure, when writing a CALL statement, that these operands are aligned accordingly in the USING phrase.

   If the called program is written in a language other than COBOL, a file-name may be specified in the USING phrase as an operand in addition to identifier.

Programming considerations

1. The program containing a CALL statement is known as the calling program. The program whose name is identified by the literal is the called program. Within a called program, the name is determined by the program-name specified in the PROGRAM-ID paragraph or in the ENTRY statement.

   When the CALL statement is executed, control is passed to the called program.

2. A called program may contain CALL statements. However, such statements may not reference the calling program, whether directly or indirectly.

3. identifiers in the USING phrase of the CALL statement point to those data items in the calling program which may be referenced by the calling program. The order in which identifiers are supplied in the USING phrase of the CALL statement, and data-names in the USING phrase of the Procedure Division header or the ENTRY statement, is significant.

   identifiers in the USING phrase of the CALL statement, and data-names in the USING phrase of the Procedure Division header or the ENTRY statement, refer in pairs to the same data according to the order in which they were specified. In other words, their correspondence is not based on their names but on the order in which they occur. An exception to this are index-names. Index-names in a calling and a called program always point to separate indices.

4. When a file-name is entered on the list of the USING phrase, the starting address of the system file control block of that file is supplied to the called program.

<u>Example</u> of the use of program communication statements

```
PROGRAM-ID.MAINPROG.
           .
           .
           .
WORKING-STORAGE SECTION.
01  A PICTURE 99.
01  B PICTURE 99.
01  G PICTURE 999.
           .
           .
           .


PROCEDURE DIVISION.
           .
           .
           .
CALL "ADDITION" USING A,B.
MOVE B TO G.
           .
           .
           .
```

```
PROGRAM-ID.ADDITION.
           .
           .
           .
WORKING-STORAGE SECTION.
           .
           .
           .
LINKAGE SECTION.
01  X PICTURE 99.
01  Y PICTURE 99.
           .
           .
           .
PROCEDURE DIVISION USING X,Y.
           .
           .
           .
ADD X TO Y.
           .
           .
           .
EXIT PROGRAM.
```

In this example, the program named MAINPROG is the calling program and the program named ADDITION is the called program.

The CALL statement

CALL "ADDITION" USING A, B.

creates the link to the standard entry point identified by the Procedure Division header. When control passes to the called program, the addresses of the identifiers A and B are associated with the data-names X and Y, respectively. The identifiers A and B are defined in the WORKING-STORAGE SECTION of the calling program, and the data-names X and Y are defined in the LINKAGE SECTION of the called program. Since the addresses of the operands are passed to the called program, the ADD statement ADD X TO Y actually adds A to B. Thus, when the EXIT PROGRAM statement passes control to the MOVE statement in the calling procedure (that is, the statement immediately following the CALL statement), the new value of B, the sum of A + B, is moved to G.

ENTRY STATEMENT

Function

The ENTRY statement is used in a COBOL subroutine (a called program) for specifying the entry point by which the subroutine may be called (as opposed to the standard entry point supplied by the Procedure Division header). Optionally, data-names may be specified, if data is to be transferred from the calling program as a parameter.

Format

```
ENTRY literal [USING data-name-1 [,data-name-2]...].
```

Rules

1. literal must be a nonnumeric literal.

   literal must be a valid program-name, i.e. it must begin with an alphabetic character, must not contain any characters other than letters and digits, and has a maximum length of 8 characters.

2. The program-name specified by the literal must be unique in the object programs which are being linked to one executable program. Also, it must not be the same as the first 8 characters of the program-name entered in the PROGRAM-ID paragraph of the program containing this ENTRY statement.

3. USING may be written only if the associated CALL statement in the calling program also contains a USING phrase. The number of operands in the corresponding USING phrases must be identical; otherwise the result will be unpredictable.

4. Each operand supplied in the USING phrase of the ENTRY statement must be defined as a data item in the LINKAGE section of the program containing this ENTRY statement; its level number must be either 01 or 77.

7

Programming considerations

1. The ENTRY statement determines the entry point in the called program. The name of the entry point is specified by the literal. A branch to this entry point is effected by a CALL statement in another program which references this entry point.

2. The USING phrase has the effect that, at execution time, data-name-1 in the ENTRY statement of the called program and data-name-1/identifier-1 in the USING phrase of the CALL statement in the calling program refer to the same set of data, which is equally available both to the called program and to the calling program. The names need not be the same; however, the number of character positions in the corresponding data descriptions must be identical. A similar relationship also exists between data-name-2, ... , as specified in the USING phrase of the ENTRY statement in the called program, and data-name-2/identifier-2, ... , as specified in the USING phrase of the CALL statement in the CALLING program. In the USING phrase of the ENTRY statement in the called program, a data-name may occur only once; in the USING phrase of the CALL statement, however, the same data-name/identifier may be specified more than once.

3. In the called program, the operands of the USING phrase are treated according to the data descriptions given in the LINKAGE section.

4. From within a program containing an ENTRY statement, the transfer of control must not lead back to that ENTRY statement.

EXIT PROGRAM STATEMENT

Function

The EXIT PROGRAM statement indicates the dynamic end of a called program.

Format

```
EXIT PROGRAM.
```

Rules

1. If an EXIT PROGRAM statement is one of several imperative-statements of a record, it must be the last statement in this record.

2. Execution of the EXIT PROGRAM statement in a called program causes a transfer of control back to the calling program. Execution of the EXIT PROGRAM statement in any program that was not called has no effect, either on program execution or otherwise.

EXIT PROGRAM STATEMENT

Function

The EXIT PROGRAM statement indicates the dynamic end of a called program.

Format

EXIT PROGRAM.

Rules

1. If an EXIT PROGRAM statement is one of several imperative-statements of a record, it must be the last statement in this record.

2. Execution of the EXIT PROGRAM statement in a called program causes a transfer of control back to the calling program. Execution of the EXIT PROGRAM statement in any program that was not called has no effect, either on program execution or otherwise.

8 REPORT WRITER

8.1 GENERAL DESCRIPTION

The specific language elements available to the programmer for generating reports (Report Writer Language Elements) enable him to give such a detailed description of a report (its format and information content) in the Data Division that a very few Procedure Division statements will be sufficient to produce that report.

A printed report shows particular information in a formatted manner. The total number of all files and report description entries in a source program must not exceed 254.

In the Data Division, the programmer names the reports that he wishes to Report Writer Control System to write, assigns these reports to a file (report file) to which all of them will generally be written later, and describes their formats as well as their information input. The statements by which the reports are generated must be supplied in the Procedure Division.

During program execution, the report is created according to the format defined; that is, usually data sums are accumulated which are then often added up to sub- and final totals; counters are incremented and decremented; each line of the page is edited and printed; in this way, the report is printed page by page. Thus, the programmer is spared the struggle with innumerable step-by-step operations.

The Report Writer implemented here is not exactly the same as in Standard COBOL, but it is functionally equivalent to the Standard Report Writer Feature, differing mainly in some details of syntax.

8.2 GENERAL DESCRIPTION OF THE DATA DIVISION

In the Data Division, the programmer must supply a file description (FD) entry for the output file to which the report is to be written, and which will also contain the name of the report (see "REPORT Clause").

As the last section of the Data Division, the programmer must add the REPORT SECTION, defining the formats for, and the information input to, each report. This section contains two types of entries:

1. **The report description** (RD). It must specify the name of the report. A page format (PAGE LIMIT clause), a character (CODE clause) preceding each line of the report (not to be printed, however), and a hierarchy of control fields (CONTROL clause) may also be supplied here.

2. **The report group description entry**. It describes, for instance, the print fields of a report group; that is, data items with line and column positioning into which data will be entered either directly (VALUE) or indirectly (sending field, sums, numbers).

The report description entry is used to specify the format of a page by defining the number of lines per page as well as the line numbers as boundaries of report group specific zones. These entries perform a control function in positioning the various report groups at the time the report is produced.

Specification of **control fields** in their hierarchical order serves the purpose of structuring the detail report groups according to hierarchical items (represented by the control fields).

Thus, the detail report groups which are functionally related to the hierarchical items are combined into several groups in the order in which they are created, in such a manner that all detail report groups within a given group are assigned the same value of the hierarchically lowest item. Depending on whether a control heading and/or control footing is defined for this level of hierarchy, each group is introduced by the control heading and is terminated by the control footing (which generally contains information applicable only to the current group, e.g. totals)-

Similarly, when another hierarchical item occurs, the above series of detail report groups is, in turn, combined into groups, now governed by the next highest item in the hierarchy, etc.

This structure is generated by the Report Writer Control System in the following way: the Report Writer Control System, before creating a detail report group, will test the control fields in their hierarchical order, from top to bottom. As soon as the Report Writer Control Systems detects a change in value, that is, a **control break**, all existing control footings are created in hierarchical order from bottom to top up to the level where the first change in value was encountered; then, all existing control headings on this level in reversed order of hierarchy; and finally, the detail report group itself.

By specifying control fields in conjunction with the report groups "control heading" and "control footing", the programmer may have this report produced in a structured format.

The **report group description entry**, formally similar to a record description, is used to describe the properties of all data items in the report. Beyond the ordinary framework of COBOL language elements for the description of data items, a report data item is assigned line and column,

i.e. a page position; in other words, the data item is declared to be a print field. Each of these fields receives information.

Three types of information may be supplied:

SOURCE information (SOURCE clause information), which is available through a data item defined outside the REPORT section.

SUM information (SUM clause information), as a result of adding up data representing, in turn, SOURCE information and/or SUM information.

VALUE information (VALUE clause information), as predefined information value.

Since a given report may contain several types of report groups, the report group description specifies the type of the report group. The seven types that may be used are shown in Table 9-1.

With its report and all associated report group description entries, the report is completely described as to its format and content (including the summation operations required); that is, all prerequisites to writing the report are met.

Type	Definition
REPORT HEADING	A report group that is written only once, as the first group of the report.
REPORT FOOTING	A report group that is written only once, as the last group of the report.
PAGE HEADING	A report group written at the top of each page. Exceptions: a) The page heading is written after the report heading. b) The page heading is suppressed when the page is explicitly reserved for the report heading or footing.
PAGE FOOTING	A report group written at the bottom of each page (exceptions: same as page heading).
DETAIL	This report group type is the only one which is not produced automatically but must be specified in a GENERATE statement. Each execution of this statement produces the specified detail report group at object time.
CONTROL HEADING	This report group is generated as a heading group for a series of detail report groups, due to a control break.
CONTROL FOOTING	This report group is generated as a footing group for a series of detail report groups, due to a control break. It usually sums up data concerning the preceding series of detail report groups.

Table 8-1 Report group types used in report group description

8.3 GENERAL DESCRIPTION OF THE PROCEDURE DIVISION

In the Procedure Division, the programmer uses the INITIATE, GENERATE, and TERMINATE statements to cause the Report Writer Control System to produce the desired report according to its full description in the Data Division.

The INITIATE statement has the effect that the Report Writer Control System performs the initialization required for writing the specified report(s). This initialization enables the Report Writer Control System to recognize, for example, whether a given GENERATE statement is the first GENERATE statement to be executed in the sequence of a program.

The GENERATE statement is responsible for creating the major part of the report. It causes a number of automatic functions to be performed in addition to generating a detail report group. For example, the control field test is carried out and a control break takes place, if necessary; that is, control footings and control headings are created. If the associated report group zone of the page does not provide space for a body group (detail report group, control footing, and control heading), then, automatically, the page footing is generated, a page change is implied, and the heading group is created. All of the functions involved in writing the individual report groups, such as incrementing or decrementing counters, supplying value, source, or sum information to the print fields, as well as positioning and writing the line(s) into which the print fields of a report group are structured, are performed automatically.

The TERMINATE statement causes the Report Writer Control System to complete the creation of the specified report(s). Thus, all control footings that belong to the report and, if present, the report footing as the last report group are written.

Within the range of the DECLARATIVES, the programmer may specify user procedures for report groups following a USE BEFORE REPORTING statement. As such procedures are executed immediately prior to the actual creation (editing, etc.) of the associated report group, this provides the programmer with a means of influencing the representation of the report group at object time.

The following four special registers are known to, and used by, the Report Writer:

LINE-COUNTER, PAGE-COUNTER, PRINT-SWITCH and CBL-CTR.

A report file is a sequentially organized file which is subject to the following restrictions:

Before entering an INITIATE statement, an OPEN OUTPUT or OPEN EXTEND statement must be performed. A TERMINATE statement must be followed by a CLOSE statement (without REEL or UNIT option). No further I-O statement may be issued for this file.

8.4 DATA DIVISION LANGUAGE ELEMENTS

8.4.1 REPORT Clause

Function

The REPORT clause relates one or several reports described in the Report Section to a file in whose file description (FD) entry this clause is specified. Each of these reports is written line by line to the file assigned by the REPORT clause and described as an output file.

Format

```
{REPORT IS  }
{           } report-name-1 [report-name-2]...
{REPORTS ARE}
```

Rule

1. Each report-name from the REPORT clause must be associated with a report description entry defining that name as a report-name.

Programming considerations

1. The name of each report to be produced by the Report Writer must appear in the REPORT clause of the file description for the sequential output file to which the report is to be written.

2. Specification of several report-names in one REPORT clause relates these reports to the file whose file description entry contains the REPORT clause. When created, these reports are written to the assigned file regardless of the order in which the report names are specified and regardless of their formats, their lengths or any similar details.

3. Each report-name defined by a report description entry must be specified in one, and only one, REPORT clause; that is, a given report may be assigned to one file only.

4. The record format (variable, fixed, or unspecified length) is determined by the entry in the RECORD CONTAINS or RECORDING MODE clause (see "Relationship between the RECORD CONTAINS Clause and the RECORDING MODE Clause").

8.4.2 REPORT SECTION

Function

The REPORT SECTION describes the format and contents of the reports to be generated.

Format

```
REPORT SECTION.
{report-description-entry {report-group-entry} ... } ...
```

Programming considerations

1. The REPORT SECTION must be the last section the Data Division.

2. The report description entry is discussed on the following pages.

3. The report group entries must all follow the related report description entry in a body.

4. A report group entry formally corresponds to a record description of the FILE or WORKING-STORAGE sections. The report group entry comprises all of the data description entries for a single report group. The first entry in the report group description begins with level-number 01. All associated data description entries begin with level-number 02 (see "Report Group Entries").

8.4.2.1 Report Description Entries

Function

The Report Description (RD) entry is used to name a report, define its page format, identify its lines, and for structuring the report. It may perform the following functions as required.

1. Specification of a character identifying the print lines of a report (CODE Clause).

2. Declaration of a group hierarchy for structuring the report (CONTROL Clause).

3. Definition of a page format by specifying vertical boundaries for report group specific zones (PAGE LIMIT clause).

Format

RD report-name

[CODE clause]

[CONTROL clause]

[PAGE-LIMIT clause]

Rules

1. RD is the level indicator, which identifies the beginning of the report description and must immediately precede the report-name.

2. The report-name must be specified in a REPORT clause of the file description entry for the file on which the report is to be written.

3. No more than 127 report and report word description entries may be specified within a report description.

4. No more than 31 data-names may be specified in a CONTROL clause within a report description.

5. No more than 127 detail report groups (DETAIL phrase in the PAGE LIMIT clause) may be specified in a report description.

6. No more than 31 control footings (FOOTING phrase in the PAGE LIMIT clause) may be specified within a report description.

7. The report-name identifies the report and, accordingly, must be unique.

The RD entry clauses are described on the pages that follow.

8

CODE CLAUSE

Function

The CODE clause specifies a character for identifying the print lines that belong to a report. This character is inserted in the first position of each print line of the report but must not be printed. It is used to separate the print lines of two or more reports that are interleaved or written consecutively on the report file.

Format

```
WITH CODE mnemonic-name
```

Rules

1. The mnemonic-name must be associated with a SPECIAL-NAMES paragraph and with only a single character (see "SPECIAL-NAMES Paragraph").

2. If a report description entry includes the CODE clause, all other report description entries assigned to the same output file (through the REPORT clause) must have a CODE clause also.

3. Associated with the mnemonic-name is the character which is written at the beginning of each print line.

4. The character for marking the print lines of a report is inserted at the beginning of the line, following the carriage control character. This character must not appear in the print format.

Programming considerations

The CODE clause must not be specified if the report is to be printed immediately ("online") or if the report file SYSLST is assigned. It must be ensured that the reports do not overlap, i.e. that during the time lapse between the INITIATE and TERMINATE statement for one report no GENERATE statement is executed for another report.

CONTROL CLAUSE

Function

The CONTROL clause defines the data items that are used as control fields of the report to determine the group hierarchy and, thus, the hierarchical structuring of the report.

Format

```
{CONTROL IS  } {data-name-1 [data-name-2]...              }
{CONTROLS ARE} {FINAL [data-name-1 [data-name-2]...]      }
```

Rules

1. No more than 31 data-names must be entered in the CONTROL clause, including the FINAL phrase if specified.

2. data-name-1, data-name-2,... may be qualified but must not be indexed.

3. No data item may be subordinated to a data-name if its length is defined in the OCCURS clause as variable.

4. data-name-1, data-name-2,... must not be defined in the REPORT SECTION but rather in the FILE or WORKING-STORAGE section. A data-name entered in the CONTROL clause may be defined in the LINKAGE section of the called program, provided that the called program is continuously available in memory from the time the report is initiated until its production is terminated.

5. The data item used may be up to 256 characters in length.

6. Each data-name must name a different data item.

   Furthermore, a data-name from the CONTROL clause must not (by redefinition or otherwise) refer to the same memory locations as another data-name from the same clause.

7. A control field is a data item specified in the CONTROL clause which is tested, whenever a GENERATE statement is executed for the same report, to ascertain whether its value has changed since the last GENERATE statement executed for that report.

   If such a change in value is found to have taken place, a "control break" results, i.e. special action (described below) will be taken before the detail group specified by the GENERATE statement is written. When a GENERATE statement is executed, changes in value will generally have occurred in several control fields. In fact, it is always the hierarchically supreme change in value to which all control break concepts (such as control break and control break level) are related.

8. FINAL, data-name-1, data-name-2, etc. define the control hierarchy of the report.

9. The data-names, in the order in which they are listed from left to right, specify the levels of the control hierarchy from major to minor. The last (i.e. rightmost) data-name is assigned the lowest level, the last but one is assigned the lowest level, the last but one is assigned the second lowest level, and so on.

10. If control heading or control footing are to be printed once only for a particular level of the hierarchy (i.e. as the first and/or last body group(s) of the report), or, to put it another way, if one heading and/or one control footing is to be written for the whole of the printed detail group of the report (e.g. for totals etc.), then these report groups (control heading and control footing) must be assigned to FINAL, i.e. the CONTROL clause must include the FINAL phrase.

11. Only data-names supplied in the CONTROL clause may be specified in RESET and TYPE clauses of the report group description entry of a report. The FINAL phrase specified in the CONTROL clause is prerequisite to using the FINAL phrase in RESET and TYPE clauses.

Programming considerations

1. The action implied by a control break depends on several things; for example, whether a control heading and/or a control footing or none of the two are defined for each control hierarchy level. If a control break occurs, the Report Writer Control System creates the following control headings and control footings (as present), in the order shown below:

   a) Control footing of the lowest level.
   b) Control footing of the next higher level.
   .
   .
   .
   c) Control footing of the level responsible for the control break.
   d) Control heading of the level responsible for the control break.
   e) Control heading of the next lower level.
   .
   .
   .
   f) Control heading of the lowest level.

   Next, the Report Writer Control System writes the detail report group initiated by the GENERATE statement.

   For example, when the control break data-names for a report are specified as YEAR, MONTH, and DAY (listed in this order in the CONTROL clause), associating each of these data-names with one control heading as well as one control footing, then, if a control break occurs for YEAR (that is, the contents of the data item YEAR has changed between two chronologically successive GENERATE statements), the body groups are printed in the following order:

   Control footing for DAY
   Control footing for MONTH
   Control footing for YEAR
   Control heading for YEAR
   Control heading for MONTH
   Control heading for DAY
   Detail report group produced by the GENERATE statement.

2. When a control break occurs at a higher level, the values in the associated control fields of each lower level are implicitly assumed to have changed, whether or not that is actually the case. For instance, if the control items MONTH and DAY have the old values MAY and 15, and the current values are JUNE and 15 (that is, a control break has occurred at MONTH level), then DAY, too, is assumed to have changed its value although the old and the new value are the same.

3. If the Report Writer Control System starts creating a body group and it detects that one of the conditions for a page change exists, it will first write the page footing (if specified), then skip to the next page, then write the page heading (if specified), and finally, it will generate the body group.

4. Creation of a detail group must be requested by the programmer from the Report Writer Control System, by supplying the appropriate GENERATE statement (to define the detail group) in the Procedure Division. All other report groups such as report heading, report footing, page heading, page footing, control headings, and control footings are written automatically by the Report Writer Control System as soon as it finds the respective conditions to be satisfied.

5. To determine a control break an alphanumeric compare is performed, regardless of how the control fields are described. This means, for example, that the Report Writer Control System will detect a change of value when the contents of a control field described with COMPUTATIONAL-3 is changed from hexadecimal "7F" to hexadecimal "7C", although both variant representations print out the positive number 7.

6. When a CONTROL clause is not specified for a report, control headings and control footings may not, and cannot, be defined for the report.

<u>Example 8-1</u>

Extract from a report:

JANUARY 14	B10	4	B	8.36	
	B10	1	C	9.00	
PURCHASES & COST FOR 1-14		5		$17.36	$136.36
JANUARY 15	B10	2	A	16.00	

The relevant report description entry includes this CONTROL clause:

```
CONTROLS ARE FINAL MONTH DAY
```

Except for the print line beginning with PURCHASES and constituting the control footing for DAY, all other print lines from the above report extract are detail groups (same report description). The control footing of DAY was written because the data changed from January 14 to January 15.

PAGE LIMIT CLAUSE

Function

The PAGE LIMIT clause is used to indicate the length of the print page and its vertical subdivision into report group specific zones.

Format

$$\underline{\text{PAGE}} \left[\begin{Bmatrix}\text{LIMIT IS}\\ \text{LIMITS ARE}\end{Bmatrix}\right] \text{integer-p} \left[\begin{Bmatrix}\text{LINE}\\ \text{LINES}\end{Bmatrix}\right]$$

$$[\underline{\text{HEADING}}\ \text{integer-h}]$$

$$[\underline{\text{FIRST}}\ \underline{\text{DETAIL}}\ \text{integer-d}]$$

$$[\underline{\text{LAST}}\ \underline{\text{DETAIL}}\ \text{integer-e}]$$

$$[\underline{\text{FOOTING}}\ \text{integer-f}]$$

Rules

1. integer-p indicates the maximum possible number of print lines per page.

2. integer-p must not exceed 999.

3. The other integers of the PAGE LIMIT clause are line numbers (as the numbering of the lines on a page starts with 1, integer-p, too, may be considered as a line number).

4. The integers of the PAGE LIMIT clause are subject to the following relationships:

   integer-h must be equal to or greater than 1.
   integer-d must be equal to or greater than integer-h.
   integer-e must be equal to or greater than integer-d.
   integer-f must be equal to or greater than integer-e.
   integer-p must be equal to or greater than integer-f.

5. integer-h of the HEADING phrase specifies the first line on the page on which anything may be written.

6. integer-h must not exceed 999.

7. integer-d of the LIMIT phrase specifies the last line on the page on which anything may be written.

8. integer-d of the FIRST DETAIL phrase specifies the lowest line number permitted for any body groups.

9. integer-d must not exceed 999.

10. integer-e of the LAST DETAIL phrase specifies the highest line number permitted for any detail groups and control headings.

11. integer-e must not exceed 999.

12. No more than 127 detail groups (DETAIL) may be specified within a report description entry.

13. integer-f of the FOOTING phrase specifies the highest line number permitted for any control footings.

14. No more than 31 control footings may be specified within a report description entry.

15. integer-f must not exceed 999.

16. if the PAGE LIMIT clause is specified but one or more of the optional integers is omitted, the following values are internally assumed by default:

Omitted integer	Value assumed when option omitted
HEADING integer-h	1
FIRST DETAIL integer-d	Value of HEADING integer-h
LAST DETAIL integer-e	Value of FOOTING integer-f
FOOTING integer-f	Value of LIMIT integer-p

17. If the PAGE LIMIT clause is omitted entirely, the compiler assumes the following values for each integer of the PAGE LIMIT clause:

Integer	Assumed value
HEADING integer-h	1
FIRST DETAIL integer-d	1
LAST DETAIL integer-e	48
FOOTING integer-f	48
LIMIT integer-p	50

Programming considerations

1. The Report Writer Control System uses the phrases of the PAGE LIMIT clause for partitioning the page into regions. Only report groups of specific types may be printed in certain regions. The page regions for each type of report group are:

   a) If the report heading description includes the NEXT GROUP NEXT PAGE clause, the report heading prints from the line whose number is integer-h through the line whose number is integer-p. Otherwise, the report heading must not print beyond the line whose number is integer-d minus 1, which requires explicit entry of integer-d (see under b. below).

   b) The page heading must print in the region from integer h through integer d minus 1. This region does no exist if the value of integer-d is established by the compiler. It is therefore necessary for integer-d to be supplied explicitly in the PAGE LIMIT clause when printing a page heading or report heading which does not appear on a page by itself.

   c) Detail groups and control headings may be printed only in the region from integer-d through integer-e, these boundaries included.

   d) Printout of control footings is permitted in the region from integer-d through integer-f.

   e) A page footing may be printed only in the region from integer-f+1 through integer-p. This region does not exist if the PAGE LIMIT clause is supplied without the integer-f option. Therefore, in this case, printing a page footing requires that integer-f be specified explicitly.

   f) If a whole page is associated with the report footing through LINE NEXT PAGE, the report footing may print in the region from integer-h through integer-p. Otherwise, report footings are subject to Rule e).

Table 8-2 is a schematic view of the partitioning of a page into regions in those cases where each integer of the PAGE LIMIT clause has a different value.

2. NEXT GROUP and LINE clauses of the report group description entries must not conflict with the PAGE LIMIT clause; that is, the lines of a report group must be within the assigned region.

	Page regions and valid entries				
integer	Region 1 REPORT HEADING and REPORT FOOTING	Region 2 PAGE HEADING and REPORT HEADING (if any)	Region 3 DETAIL and CONTROL HEADING	Region 4 CONTROL FOOTING	Region 5 PAGE FOOTING and REPORT FOOTING (if any)
integer-h	....	....	....	....	....
integer-d	....	1)	....	....	....
integer-e	....	....	....	....	
integer-f	....	....	....	....	2) ....
integer-p	....	....	....	....	....

Table 8-2 Page partitioning into regions (schematic)

1) Region 2 is through integer-d -1.

2) Region 5 starts at integer-f +1.

Table 8-2 shows all print page regions which result when the PAGE LIMIT clause is specified with integer-h less than integer-d less than integer-e less than integer-f less than integer-p. The same assumption applies to the following rules for using these regions.

Region 1:

Scope: integer-h through integer-p.

Contents: report heading or report footing.

Rules: If the report heading description includes the NEXT GROUP clause with the NEXT PAGE phrase, a whole print page is reserved for the report heading; that is, no other report group is printed on this page. Whether the report head is printed using the whole or any part of the region, is determined by the LINE clause of the report heading description.

If the report footing description includes the first (or only) LINE clauses with the NEXT PAGE phrase, a whole print page is reserved for the report footing; that is, the same rules apply as to the report heading.

Region 2:

Scope: integer-h through the line immediately preceding integer-d.

Contents: report heading and page heading, or page heading only.

Rules: The report heading will be printed in region 2 only if its report group description does not include the NEXT GROUP PAGE clause. Since a report heading is printed only once per report, the report heading appears in region 2 only on the first page of the report.

With the exception of those pages which are exclusively reserved for the report heading and report footing, a page heading will appear in region 2 on all pages (if a page heading is supplied at all).

If the first page is provided with both the report heading and the page heading, the page heading must be printed after the report heading (LINE and NEXT GROUP clauses of the two report description entries must not conflict with this rule).

Region 3:

Scope: integer-d through integer-e.

Contents: body group.

Rules: As shown by the scope, region 4 generally extends beyond region 3 at the bottom.

Any body group may be written in region 3 or in the overlapped portion of region 4. The remain portion of region 4 may receive control footings only.

Region 4:

Scope: integer-d through integer-f.

Contents: body group.

Rules: As shown by the scope, region 4 generally extends beyond region 3 at the bottom.

Any body group may be written in region 3 or in the overlapped portion of region 4. The remain portion of region 4 may receive control footings only.

Irrespective of the body's group type, region 4 applies to the NEXT GROUP clause of this group.

Region 5:

Scope: integer-f+1 through integer-p.

Contents: report footing and page footing.

Rules: If the first LINE clause in the report description entry for the report footing does not contain the NEXT PAGE option, the report footing will be printed in region 5 on the last page of the report.

The page footing of a report is always written in region 5.

If both the page footing and the report footing are to be printed in region 5 of the last page, then the page footing must precede the report footing. The LINE clauses of the two report groups must not be in conflict with this rule.

8.4.2.2 Report Group Entries

Function

The report group entry describes and defines the format, type, and properties of a report group as a series of elementary items and items associated with information. These items, organized in lines and columns, form the line(s) to be printed in a report group.

A prepositioning to the next report group to be written may be achieved by a NEXT GROUP phrase.

Using the type of report group, demands are placed both on the description of the report group and on the Report Writer Control System when the report group is created. For example, the page positioning of a report group is also determined by its type. Much the same is true of the time the report group is produced. Summations, for instance, can be defined in control footings only. They are executed immediately preceding the creation of detail groups or the control footings concerned, depending on type of summation.

Format

```
01 [data-name-1]
    TYPE clause
   [LINE clause]

   [NEXT GROUP clause]

02 [data-name-2]

   [LINE clause]

   [COLUMN clause]

   [GROUP INDICATE clause]
    PICTURE clause
   [BLANK WHEN ZERO clause]

   [JUSTIFIED RIGHT clause]

   ⎧SOURCE clause⎫
   ⎨SUM clause   ⎬
   ⎩VALUE clause ⎭
```

Rules

1. The description of a report group must begin with a 01-level entry, followed b at least one 02-level entry.

2. With the exception of the data-name, which must immediately follow the level-number, the clauses may be supplied in any order.

3. A report consists of a series of report groups printed on one or more pages. Up to seven different report group types may appear in the report (see "TYPE Clause").

   By definition, a report group consists of one or several elementary items, where each item with a COLUMN clause (printable item) must be assigned to one line. Report groups which do not contain printable items are nonprintable. A printable report group is a unit comprising one or more lines.

   With the exception of the report heading and the report footing, any report group may be created several times in the course of creating report; in this case, there must ben o change in the report group format or in the constant information contents.

4. The data-name which immediately follows the level-number 01 is the name of the report group. It must be supplied if the report group is to be referenced directly in a GENERATE statement (detail group), a USE BEFORE REPORTING statement, an UPON phrase within a SUM clause to be used for qualification.

5. The TYPE clause specifies the type of report group, enabling the Report Writer Control System to determine the modalities (when, where) for producing that report group.

6. The LINE clause relates a line of the current or the next print page to the printable items specified within its range (until the next LINE clause or the end of the report group entry). Therefore, the first LINE clause of a report group description entry determines the positioning of that report group.

7. The NEXT GROUP clause causes (after one report group is printed) the next report group to be positioned to the line number supplied in the phrases of the NEXT GROUP clause.

   The COLUMN, GROUP INDICATE, BLANK, JUSTIFIED, and PICTURE clauses define the location and format of the printable data item for a particular print line of the report group.

8. SOURCE, SUM, and VALUE clauses associate the data items with the respective information. In addition, the SUM clause also defines an internal sum counter whose contents is automatically updated by the Report Writer Control System.

9. Each printable item must be covered by a LINE clause. This LINE clause represents a print line.

8

Programming considerations

1. The name of a report group may be specified in the REPORT SECTION and in the Procedure Division. Only names of control footings and detail groups may be supplied in the REPORT SECTION. While the name of a detail group may be defined in the UPON phrase of a SUM clause, the name of a control footing may be used only for qualifying a sum counter. In the Procedure Division, the name of a report group may be specified only in two cases:

   a) The name of a detail group may be specified in a GENERATE statement.

   b) In a USE BEFORE REPORTING statement, any report group name (except names of detail groups) may be used.

   As a qualifier for a report group name, only the associated report name is allowable.

2. A data-name immediately following the level-number 02 of a report group entry should not be specified unless that data item description includes a SUM clause. In this case, the data-name will be interpreted as the name of the sum counter which is established as a result of the SUM clause (see "SUM clause"). The name of a sum counter may again be specified in a SUM clause. If the data-name is supplied although no SUM clause is present, in other words, if the data-name is defined as the name of the item, its use is illegal.

   Sum counters may only be qualified by report group names and/or report names.

Report group entry clause summary

Table 8-3 lists the report group entry clauses and briefly describes each.

The BLANK WHEN ZERO, JUSTIFIED, PICTURE, and VALUE clauses are also used in other sections of the Data Division; that is, they are assumed to be known here (refer to the pertinent descriptions).

All other report entry clauses are described on the following pages, in alphabetical order.

Clause	Brief description
BLANK WHEN ZERO	Indicates that a printable item is not to be printed when its contents are zero.
COLUMN	Defines an item as printable by specifying the starting position (column number) of that field on the print line.
GROUP INDICATE	Indicates that the printable item is to be printed only if the detail group with which it is associated is written for the first time in the report, after a page advance or a control break.
JUSTIFIED	Explicitly specifies the positioning of data in the associated item.
LINE	Specifies the line on the report page on which one or more printable items within its range are to be printed.
NEXT GROUP	Specifies prepositioning for the next report group to be printed.
PICTURE	Specifies the characteristics of a data item and an internal sum counter.
SOURCE	Associates a data item with a sending field as its source of information.
SUM	Indicates how and which addends are to be accumulated in the internal sum counter. The sum counter is built automatically. Also, this clause associates the data item with the sum counter as its source of information.
TYPE	Specifies the particular type of report group being described.
VALUE	Defines a constant value for a printable item.

Table 8-3 Report group entry clauses

8

COLUMN CLAUSE

Function

The COLUMN clause defines a data item as a printable field by specifying its column number (integer) to indicate the starting position of that item with respect to the print line.

Format

COLUMN integer

Rules

1. integer must be an unsigned integer in the range of 1 ≤ integer ≤ 251.

2. The column number (integer) specifies the position in which the first (leftmost) character position of the printable item is to appear on the print line. The first (leftmost) printable character position of the print line is considered to be column 1. (The highest column number depends on the type of printer being used.)

Programming considerations

1. The presence of the COLUMN clause in the description entry for a data item containing the mandatory PICTURE clause as well as either a SOURCE, SUM, or VALUE clause constitutes a printable item.

2. Every entry that defines a printable item must either contain, or be preceded by (in the same report group description), a LINE clause. The printable item will be printed on the print line specified by the LINE clause.

3. Within the scope of a LINE clause (that is, before the next LINE clause occurs or until the report the report entry ends), the printable items must be defined in ascending column number sequence. The printable items are printed on the line in the order in which they were defined.

4. Printable items on a print line must not be so defined that they overlap each other (concerning the COLUMN and PICTURE clauses).

5. Immediately before the printable items in a line are printed, i.e. before the print line is written to the report file, the information is automatically moved to these items by implicit MOVE statements. Depending on whether the description of the receiving item (= printable item) includes the SOURCE, VALUE, or SUM clause, the sending item will be the identifier from the SOURCE clause, the literal from the VALUE clause, or the internal sum counter which was established for the SUM clause.

6. The Report Writer Control System supplies space characters for all positions of a print line that are not occupied by printable items.

Example 8-2

Assume the following description of an elementary item in a report group entry:

```
02 LINE 3 COLUMN 30 PIC A(12) VALUE IS "EXPENDITURES"
```

When this report group is produced, the character string EXPENDITURES will be printed on line 3 of the current report page, starting at column 30.

GROUP INDICATE CLAUSE

Function

The GROUP INDICATE clause indicates to the Report Writer Control System that the printable item with which it is associated is to be printed only if the detail group is written for the first time on a page of the report or for the first time after a control break.

Format

GROUP INDICATE

Rules

1. The GROUP INDICATE clause may be used only in a detail report group. The field description containing the GROUP INDICATE clause must also include a COLUMN clause.

2. The GROUP INDICATE clause causes the Report Writer Control System to print the associated printable item according to the SOURCE or VALUE clause specified, only in the following cases:

   a) in the report
   b) after a page advance or
   c) after a control break.

   In all other cases, space characters are substituted for the printable field, suppressing the associated data in the print layout.

Example 8-3

a) Data Division entries:

```
              .
              .
              .
     CONTROLS ARE FINAL, MONTH, DAY
              .
              .
              .
  01 DETAIL-LINE PLUS 1 TYPE DETAIL.
     02 COLUMN 2 GROUP INDICATE PIC A(9)
        SOURCE MONTHNAME (MONTH).
     02 COLUMN 13 GROUP INDICATE PIC 99 SOURCE DAY.
              .
              .
              .
```

b) Excerpt from a report

```
         .
         .
         .
    JANUARY 15 B11 16 A 20.00
               B11  4 B 13.45
               B11 20 D 35.40
```

The above extract from a report comprises three detail groups of the same report group definition, created immediately following a control break (change in the current contents of DAY).The current values JANUARY and 15 of the sending items (SOURCE fields) for the first two printable items of this 1-line detail report group appear, therefore, only in the first line (detail group).

LINE CLAUSE

Function

The LINE clause specifies the line on the report page on which one or more printable items defined within its scope (see COLUMN clause) are to be printed. In special application, it is used simply for page advance.

Format

$$\underline{\text{LINE}}\ \text{NUMBER IS} \begin{Bmatrix} \text{integer-1} \\ \underline{\text{PLUS}}\ \text{integer-2} \\ \underline{\text{NEXT}}\ \underline{\text{PAGE}} \end{Bmatrix}$$

Rules

1. integer-1 and integer-2 may only be specified as unsigned integers equal to or greater than 1.

2. A LINE clause with integer-1 phrase specified is called an absolute LINE clause. Integer-1 is interpreted as a line number. Therefore, when creating the associated report group, integer-1 indicates the particular line on the report page on which the printable items which belong to the LINE clause and, thus, are organized as a line will ultimately be written. In this way, an absolute LINE clause always defines a print line.

3. A LINE clause with PLUS integer-2 phrase specified is called a relative LINE clause. The print line to receive the printable items associated with a relative LINE clause at the time the particular report group is generated, is determined relative to the last vertical positioning. For this purpose, integer-2 is added to the number of the line on which the report page is presently positioned (see "NEXT GROUP Clause" and "LINE COUNTER Special Register"). The sum then designates the number of the next line to be printed.

   Deviations from the above rules are only permissible for special types of report groups, when the relative LINE clause was used for the first line of a report group. These deviations will be discussed in due course.

4. A LINE clause with NEXT PAGE phrase causes the associated report group to be printed on a free page (usually the next page), advancing the page before the report group is printed.

   Whether a LINE NEXT PAGE clause additionally specifies a print line can be seen from the rules described below.

Programming considerations

The following programming considerations are classified into two categories - General Notes and Report Group Notes. Rules in the latter category describe the use of the LINE clause in the descriptions of the different types of report groups.

The abbreviations used in the Report Group Notes have the following meaning.

Abbreviation	Meaning
A	one or more absolute LINE clauses
R	one or more relative LINE clauses
NP	a LINE clause with NEXT PAGE phrase

General notes

1. A printable report group item that is to be written on a line of the report page must either itself contain a LINE clause in its description or its description must be preceded, within the associated report group description entry, by a LINE clause. Conversely, all printable items in the associated report group description entry, which follow the LINE clause and precede the next LINE clause or the end of that report group description entry, will be written on the line determined by the LINE clause. Subsequent printable items from the report group entry are printed combined into lines in the same manner.

2. The vertical spacing parameters (LINE and NEXT PAGE clauses) must be selected for the different report group types in such a way that these report groups can be printed within the page regions provided for them (see also "PAGE and TYPE clauses"). It is not possible to continue a a report group in the associated specific region on another page.

   If a body group cannot be printed in its specific region simply because part of that region is no longer available, then that body group as a whole will be written in the same region on the next page, after creating first the page footing on the old page and the page heading on the new page.

3. If a report group entry includes (one or more) absolute LINE clauses, they must all be specified:

   a) preceding the first relative LINE clause if any, and
   b) in the order of ascending integers.

4. In each report group entry, the LINE NEXT PAGE clause may only be specified as the first LINE clause; thus, it may only be written once.

Report group notes

5. Report heading

The following sequences of LINE clauses may be used in describing a report heading group:

$$\left\{\begin{array}{l} A \\ A\ R \end{array}\right\}$$

6. Page heading

Only the following sequences of LINES clauses may be used in describing a page heading group:

$$\left\{\begin{array}{l} A \\ A\ R \\ R \end{array}\right\}$$

When the first LINE clause from the description of the page heading is a relative clause, then, generally, the first line of the report group is printed relative to the region boundary integer-h of the PAGE LIMIT clause (see "PAGE LIMIT Clause"). Only when the report heading was printed on the same page will there be deviation in that the first line of the report group is printed relative to the vertical spacing resulting from the creation of the report group heading.

7. Body groups

Detail groups, control headings, and control footings are referred to generically as body groups.

a) The following sequences of LINE clauses may be used in describing a body group:

$$\left\{\begin{array}{l} NP \\ NP\ A \\ NP\ A\ R \\ NP\ P \\ A \\ A\ R \\ R \end{array}\right\}$$

b) When NP A or NP A R sequences are specified for a LINE NEXT PAGE clause, NP provides page changing instructions, and does not define a print line. Thus, the first absolute LINE clause must appear with, or prior to, the first entry defining a printable item.

c) If a LINE clause with NEXT PAGE phrase is specified as the only LINE clause or with the sequence NP R, this implies not only page advance but also vertical spacing of the first line of that report group.

d) A LINE NEXT PAGE clause in the description of a body group indicates to the Report Writer Control System that the body group is to be printed on the next page which is not yet occupied by a body group. This may lead to a situation where no page advance is carried out. This will certainly be the case when the body group is to be written as the first body group on a particular report.

e) If a body group defined with a LINE clause sequence NP, NP R, or R is the first body group to be printed on a page, then the first print line of that body group will be printed on the line number designated by integer-d as supplied in the PAGE clause. However, it may well be the case that the paper has already been advanced beyond region boundary integer-d, e.g. through the last body group printed because of a NEXT GROUP clause (see under that clause). In this case, the first line of the body group will be written on the line immediately following the present positioning.

f) If a body group whose first LINE clause is relative is not to be printed as the first body group on a page, the present positioning (line number) is advanced by the number of lines specified by, integer-2 in the first LINE clause, to the new position where the first line of this report group will be printed.

8. Page footing

Only the following sequences of LINE clauses may be used in describing a page footing:

$$\left\{\begin{matrix} \text{A} \\ \text{A R} \end{matrix}\right\}$$

9. Report footing

a) The following LINE clause sequence alternatives may be used in describing a report footing:

$$\left\{\begin{matrix} \text{NP A} \\ \text{NP A R} \\ \text{A} \\ \text{A R} \\ \text{R} \end{matrix}\right\}$$

b) The LINE clause with the NEXT PAGE phrase specified is used solely for page advance.Therefore, the first absolute LINE clause must appear with, or prior to, the description of the first printable item, in order to permit positioning to the first line of the report group.

<u>Example 8-4</u>

```
01  DETAIL-GROUP TYPE DETAIL LINE NEXT PAGE.
    02  LINE 10 COLUMN 1  PIC X(10) VALUE "1ST ITEM".
    02          COLUMN 15 PIC X(4)  SOURCE CARD-FIELD-1.
    02  LINE 12 COLUMN 1  PIC X(10) VALUE "2ND ITEM".
    02          COLUMN 15 PIC 9(5)  SOURCE-WORK-FIELD-1.
```

The two-line detail group is printed on lines 10 and 12 of a page not yet used for other body groups, because the LINE clause sequence is NP A.

Example 8-5

```
01  DETAIL-LINE LINE PLUS 1 TYPE DETAIL.
    02  COLUMN 2 GROUP INDICATE PIC A(9) SOURCE FIELD-NO-1.
```

This example shows a report group whose description includes only one relative LINE clause. The report group will be printed on the line resulting from the present position being advanced by one line. If the report group is to be printed as the first body group on that page, and the present positioning has not gone beyond the integer-d region boundary (FIRST DETAIL phrase if the PAGE LIMIT clause), the report group is printed to the line whose number is integer-d.

NEXT GROUP CLAUSE

Function

The NEXT GROUP clause indicates to the Report Writer Control System the particular line on a page to which the paper is to advance after printing the last line of the report group containing the NEXT GROUP clause, thereby causing the paper to be prepositioned for the next (chronologically-speaking) report group to be printed (minimum distance from the next report group).

Format

```
                ⎧integer-1      ⎫
NEXT GROUP IS   ⎨PLUS integer-2 ⎬
                ⎩NEXT PAGE      ⎭
```

Rules

1. integer-1 and integer-2 must be specified as unsigned integers equal to or greater than 1. Their value must not exceed 999.

2. The NEXT GROUP clause must not be specified in a report group description entry unless a LINE clause has been specified.

3. A NEXT GROUP clause with integer-1 phrase is called an absolute NEXT GROUP clause. After printing the last line of the associated report group, the Report Writer Control System advances the paper to the line whose number is specified by integer-1. This may also involve a page advance, printing the page footing on the previous page and the page heading on the new page (for body groups only).

4. A NEXT GROUP clause with PLUS integer-2 phrase specified is called a relative NEXT GROUP clause. Positioning is advanced by integer-2 lines from the last print line of the associated report group.

5. A NEXT GROUP clause with NEXT PAGE phrase specified indicates that the current page is considered to be full; that is, after the associated report group is printed, a page advance is executed (automatic generation of page footing and page heading for body groups).

6. If in a USE BEFORE REPORTING procedure of a control footing, the PRINT SWITCH special register is set to 1 by a MOVE statement, the function of the NEXT GROUP clause is suppressed by the Report Writer Control System.

8

Programming considerations

1. The NEXT GROUP clause may appear only in a 01-level report group description entry.

2. Rules for the report heading

   a) If the report heading is to appear on a page by itself, its description must contain the NEXT GROUP clause with the NEXT PAGE phrase.

   b) If the report heading is not to appear on a page by itself, the following rules must be observed:

      - The NEXT GROUP clause must not include the NEXT PAGE phrase.

      - integer-1 of an absolute NEXT GROUP clause must indicate a line number greater than the one assigned to the last print line of the report heading.

      - An absolute or relative NEXT GROUP clause must be selected in such a way that the page heading as the next report group can still be contained in its specific region. A positioning to the line with the number integer-d (see "PAGE LIMIT Clause") or even to a line which has a higher number owing to the NEXT GROUP clause is illegal.

3. Rule for the page heading

   The NEXT GROUP clause must not be used in describing the page heading.

4. Rules for body groups

   a) integer-1 of an absolute NEXT GROUP clause must be greater than or equal to integer-d yet less than or equal to integer-f (see "PAGE LIMIT Clause").

      If the line number integer-1 of the absolute NEXT GROUP clause is less than or equal to the number of the line was printed as the last line of the body group whose description includes the NEXT GROUP clause, the Report Writer Control System executes a page advance (automatic generation of page footing and page heading included) and positions on the line specified by integer-1.

   b) If a relative NEXT GROUP clause would advance the paper beyond the lower boundary specified by integer-f (see "PAGE LIMIT Clause"), the Report Writer Control System executes a page advance (generation of page footing and page heading included) and positions on the upper region boundary specified by integer-d (see "PAGE LIMIT Clause").

   c) The NEXT GROUP clause with the NEXT PAGE phrase specified indicates that no further body group is to be written on the current page. The Report Writer Control System positions on the upper region boundary integer-d of the next page (including the creation of page footing and page heading).

   d) If several control footings are created in immediate succession as a result of a control break, the Report Writer Control System can be instructed to execute the function of the NEXT GROUP clause only for the particular control footing associated with the level of hierarchy at which the control break occurred (see "CBL-CTR Special Register").

5. Rule for the page footing

   The NEXT GROUP clause must not be used in describing the page footing.

6. Rule for the report footing

   The NEXT GROUP clause must not be used in describing the report footing.

<u>Example 8-6</u>

```
01  LINE PLUS 2 NEXT GROUP PLUS 1.
    TYPE CONTROL FOOTING DAY.
    02...
```

The above NEXT GROUP clause causes the Report Writer Control System to advance only 1 line after the above control footing is created.

SOURCE CLAUSE

Function

The SOURCE clause specifies the data item whose contents the Report Writer Control System (by an implicit MOVE statement) will move to the printable item whose description includes the SOURCE clause, when this item is to be printed.

Format

SOURCE IS identifier

Rules

Any identifier defined in any section of the Data Division may be specified in a SOURCE clause. In the case of the REPORT section, however, a SOURCE clause may use only the PAGE-COUNTER special register, the LINE-COUNTER special register, or a sum counter if it is associated with the report in whose description the SOURCE clause appears.

Programming considerations

1. The description of an elementary item containing a SOURCE clause must also contain a COLUMN clause, that is, the field must be printable.

2. The SOURCE clause in conjunction with the COLUMN clause generates an implicit MOVE statement. For this MOVE statement, the source (or sending) field is defined by the identifier from the SOURCE clause. The receiving field is the printable item whose description includes the SOURCE clause. The picture strings of the PICTURE clauses of the two fields must adhere to the rules of the MOVE statement (see "MOVE Statement").

3. As the SOURCE clause can never change the value of the data item it specifies, it may also refer to a control field. Generally, the system prints the value of the sending field as it is when the MOVE statement is executed (creation of the associated report group). In control footings, which are created just after the contents of the associated control data items have changed, it is often desirable to retain the previous values of the control items since they represent footing groups for the series of detail groups created in the period during which the old contents of the control items remained the same. Function 1 of the CBL-CTR special register answers this requirement (see "CBL-CTR").

4. In complicated and elaborately designed reports, the programmer may wish to print report group headings and report group footings (excluding page heading and page footing) which contain values characteristic of the control break. For further details, refer to the "CBL-CTR Special Register" discussion.

Example 8-7

```
FILE SECTION.
     ...
     02 DEBT  PIC XXX.
     ...
REPORT SECTION.
     ...
     02 COLUMN 19 PIC XXX SOURCE DEBT.
     ...
```

The SOURCE clause shown in this example indicates that the value in the data item DEPT is to be moved to the printable item concerned, when the associated report group is printed.

SUM CLAUSE

Function

The SUM clause instructs the Report Writer Control System to create and (immediately or subsequently) print arithmetic sums of data items selected from the detailed information that constitutes the report. The SUM clause indicates to the Report Writer Control System the addends which are to be used for summation. In addition, this clause provides a numeric item, which is generated automatically, for the accumulation of the addends. This field, i.e. the sum counter, is also the sending field (source item) for the implicit MOVE statement used to edit the printable field whose description contains the SUM clause.

Format

```
SUM identifier-1 [identifier-2] ... [UPON data-name-1]

     [          {identifier-3}]
     [RESET ON  {            }]
     [          {FINAL       }]
```

Rules

1. An identifier used as an addend in a SUM clause must be defined in the FILE, WORKING-STORAGE, LINKAGE, or REPORT SECTION. From the REPORT SECTION, only the name of a sum counter may be referenced as an addend of a SUM clause.

2. Each identifier used as an addend in a SUM clause must represent a numeric data item.

3. data-name-1 is permitted only as the name of a detail group that is defined in the current report description.

4. FINAL or identifier-3 must appear in the CONTROL clause of the current report description.

5. identifier-1, identifier-2, and all corresponding identifiers specify the items to be accumulated in the sum counter.

6. The UPON phrase has the effect that the specified addends are added up only when a GENERATE statement is executed referencing the very detail group defined in the UPON phrase (see "Use of UPON phrase", below).

7. The RESET phrase overrides the standard reset function of the sum counter to zero (see "Use of RESET phrase").

Programming considerations

1. The SUM clause can only be specified in control hierarchies.

   Assume the following excerpt from a report:

```
JANUARY 02 B10                    2 A    3.00
           B12                    1 A    4.00
           B12                    3 C   17.00
PURCHASES & COST FOR 1-02         6    $24.00
```

   Printed here are the detailed cost figures and their sum for January 2.

   The detailed cost figures shown on the first three print lines were printed as a result of a GENERATE statement referring to the following detail group:

```
01  DETAIL-LINE TYPE DETAIL.
    02  ...
    .
    .
    .
    02  COLUMN 50 PICTURE ZZ9.99 SOURCE COST.
```

   This GENERATE statement was executed three times without control break, causing the detail group to be printed three times in succession. The data item by the name of COST, described in the FILE SECTION of the Data Division as:

```
02 COST PICTURE 999V99.
```

   supplied the detailed cost figures.

   When that GENERATE statement was executed once more, the value of the control data item DAY had been changed from 2 to a different value. The following control footing:

```
01  ... TYPE CONTROL FOOTING DAY.
    02  ...
    .
    .
    .
    02  SUM-DAY COLUMN 49 PICTURE $$$9.99 SUM COST.
```

   was generated as the fourth line of the above partial report, with the contents of sum counter SUM-DAY printed starting at column 49.

   Summation took place as follows:
   During compilation, the compiler created a sum counter (named SUM-DAY) in response to the SUM clause. At object time, when executing the appropriate INITIATE statement, the Report Writer Control System reset the sum counter to zero. Then, each time the GENERATE DETAIL-LINE statement was executed, the current value of the data item COST was added to the contents of the SUM-DAY counter. Since the Report Writer Control System performs this summation (see "Detail Summation") immediately following the control break test and the action resulting from that test, the sum of the cost figures for January 2 is printed out with the control group footing for DAY. When the control footings are created, the sum counter is finally reset to zero, as the SUM clause contains no RESET phrase.

2. Use of the PICTURE clause

   If the description of a data item, within a given report group entry, includes a SUM clause, the associated PICTURE clause describes not only the data item but also the sum counter which the compiler will establish due to the SUM clause. The data item, if printable, is used for printing out the contents of the associated sum counter. The PICTURE clause must define the data item as a numeric or numeric-edited data item, where editing symbols for sum counters will be ignored.

3. Use of the sum counter

   if the data item entry that contains a SUM clause has a data-name immediately following the level-number, that data-name is the name of the internal sum counter (for example, for rounding its contents prior to printing). The internal sum counter is based on COMPUTATIONAL-3 if all addends from the present SUM clause are identified as decimal operands. The capacity of the sum counter, both integral and fractional (to the right of the decimal point), will be equivalent to that specified by the PICTURE clause.

4. Types of summation

   The programmer can specify three types of summation: detail-incrementing, rolling forward, and crossfooting.

5. Detail-incrementing

The time at which the Report Writer Control System adds up an addend (identifier-1 or identifier-2 from the SUM clause) in the related sum counter depends on the addend itself.

The addends used for detail-incrementing are those which are not themselves sum counters, in other words, are defined outside the REPORT section.

Detail-incrementing is the basis for the other two types of summation. The term "detail-incrementing" derives from the fact that typically the addends involved in it are printed with the detail groups of the report.

Detail-incrementing occurs each time that GENERATE statements are executed. Therefore, the programmer must ensure that the operands used for detail-incrementing contain the required values at the time that GENERATE statements are executed. If a SUM clause uses the UPON phrase, the addends in that SUM clause are added into their sum counter only when this detail-incrementing operation takes place in executing a GENERATE statement referring to the same detail group as the UPON phrase (there is, therefore, no point in using the UPON phrase for a summary report). However, if the SUM clause does not include the UPON phrase, then those addends which are not defined as sum counters are added to their related sum counters when any GENERATE statement for the report is executed (detail-incrementing).

The Report Writer Control System performs detail-incrementing only after taking certain actions as regards control break (test; creation of the control footings and headings if test is positive). This control break processing also includes resetting the sum counters to zero after creating the control footing whose description contains the corresponding SUM clauses (see "RESET Clause"). This ensures that the printed sum will contain only the values of the addends for the particular series of detail groups which is concluded by the associated control footing (for example, the sum of the cost figures for January 2).

6. Rolling-forward (hierarchical summation)

The prerequisite for this kind of summation is that a SUM clause of a control footing must specify as an addend at least one sum counter which was defined as the result of a SUM clause of a hierarchically lower (that is, less inclusive) control footing. Therefore, rolling-forward cannot be designated unless a report description includes at least two control footings, whose descriptions each contain at least one SUM clause.

The contents of a sum counter which is specified as an addend in the SUM clause of another control footing will be added, at the time the associated (hierarchically lower) control footing is generated, to the contents of the sum counter in whose SUM clause it appears as an addend.

The following example illustrates rolling forward:

Example 8-8

```
01  ... TYPE CONTROL FOOTING MONTH.
    02  ...
    .
    .
    .
    02  SUM-MONTH COLUMN 46 PICTURE $$$9.99 SUM
        SUM-DAY.
```

In the above control footing description, the rolling-forward function is specified in conjunction with the following control footing description (see Example 8-7):

```
01  ... TYPE CONTROL FOOTING DAY.
    02  ...
    .
    .
    .
    02  SUM-DAY COLUMN 49 PICTURE $$$9.99 SUM COST.
```

For each creation of the control footing with the control item DAY, the Report Writer Control System adds the contents of the sum counter SUM-DAY to the contents of the sum counter SUM-MONTH, before resetting SUM-DAY to zero. If either a control break or execution of a TERMINATE statement causes the control footing to be generated with MONTH, the sum counter SUM-MONTH (before resetting to zero) will contain the sum of all day-sums (values of SUM-DAY at summation times) of the current month.

7. Crossfooting (adding hierarchically equal sums)

This type of summation takes place when a SUM clause contains, as addends, the names of sum counters defined by other SUM clauses in the same control footing. Normally, such addends are sum counters whose values are created through detail-incrementing.

Example 8-9

```
01  MINOR TYPE CONTROL FOOTING...
    02  SUM-1 SUM WORKING-ITEM-1...
    02  SUM-2 SUM WORKING-ITEM-2...
    02  SUM   SUM SUM-1 SUM-2...
```

WORKING-ITEM-1 and WORKING-ITEM-2 are data items defined in the WORKING-STORAGE section of the Data Division. Sum counter SUM accumulates the values of SUM-1 and SUM-2, previously generated through detail-incrementing.

The Report Writer Control System performs crossfooting just before printing the control footing concerned. If more than one SUM clause requires such addition, the order of execution is determined by the sequence of these SUM clauses. This order is essential to the result of the addition.

Obviously, this type of addition is carried out before rolling-forward, thereby ensuring that sums hierarchically equivalent in summation may also be rolled forward.

Example 8-10

```
                .
                .
                .
    CONTROLS ARE STATE, CITY.
                .
                .
                .
01  LINE PLUS 2 TYPE CONTROL FOOTING CITY.
    02  SUM-1 SUM MALES...
    02  SUM-2 SUM FEMALES...
    02  SUM-CITY SUM SUM-1, SUM-2...
01  LINE PLUS 1 TYPE CONTROL FOOTING STATE.
    02  SUM-STATE SUM SUM-CITY...
                .
                .
                .
```

The values accumulated in sum counter SUM-CITY by crossfooting the values from the hierarchically equivalent sum counters SUM-1 and SUM-2 (detail-incrementing) are rolled forward in sum counter SUM-STATE (the control footing with STATE is higher in hierarchy than the control footing with CITY). This is possible only because the sum counter SUM-CITY contains the proper value before rolling-forward takes place in the sum counter SUM-STATE.

8. Mixing operands

   A SUM clause that does not contain an UPON phrase may include one or more of each of the following kinds of operands (= addends):

   a) Operands defined in the FILE, WORKING-STORAGE, and LINKAGE Sections.

   b) Operands defined as sum counters in a hierarchically inferior control footing.

   c) Operands defined as sum counters in the same control footing (whose description contains the SUM clause).

   Summing for each of the above kinds of operands occurs at the times indicated in the preceding discussions.

9. Use of the UPON phrase

   a) When an UPON phrase is used in a SUM clause, all addends of that clause must be defined outside the REPORT section; that is, they may be defined only in the FILE, WORKING-STORAGE, and LINKAGE sections.

   b) The UPON phrase has the effect of preventing detail-incrementing of the addends from the present SUM clause, unless a GENERATE statement is executed specifying the detail group indicating in the UPON phrase. A detail-incrementing caused by any other GENERATE statement will not, therefore, affect any of the addends in the SUM clause in question.

Example 8-11

```
DATA DIVISION.
FILE SECTION.
FD  INFILE...
    .
    .
    .
    RECORDS ARE MUELLER, MEIER.
01  MUELLER PICTURE 999.
01  MEIER PICTURE 9999.
REPORT SECTION.
    .
    .
    .
01  MUELLER-DETAIL TYPE DETAIL.
    02  LINE PLUS 1 COLUMN 1 PIC 999 SOURCE MUELLER.
01  MEIER-DETAIL TYPE DETAIL.
    02  LINE PLUS 1 COLUMN 1 PIC 9999 SOURCE MEIER.
    .
    .
    .
01  MINOR TYPE CONTROL FOOTING...
    02  SUMME-1 SUM MUELLER UPON MUELLER-DETAIL...
    02  SUMME-2 SUM MEIER UPON MEIER-DETAIL...
    .
    .
    .

PROCEDURE DIVISION.

    GENERATE MUELLER-DETAIL.
    .
    .
    .
    GENERATE MEIER-DETAIL.
```

Because MUELLER and MEIER are the names of two different data records on the same file, they cannot be available in memory concurrently. When a MUELLER record is read, the statement GENERATE MUELLER-DETAIL is executed; at that time, the current value of MUELLER is added to sum counter SUM-1. The present value of MEIER, on the other hand, is not added to the contents of sum counter SUM-2 at this time. When a MAIER record is read, the statement GENERATE MEIER-DETAIL is executed; at this time, the detail-incrementing occurs in sum counter SUM-2, and not in SUM-1.

10. Use of the RESET phrase

a) Only a data-name (FINAL included) supplied in the CONTROL clause of the same report may be used in a RESET phrase. Moreover, the control field must be at a higher level in hierarchy than the control footing whose description contains the RESET phrase.

b) Normally, the Report Writer Control System resets a sum counter to zero immediately after printing the control footing in whose description it is defined. A sum counter whose SUM clause contains the RESET phrase will be reset to zero only at a time when the (explicit or implicit) control footing for the control field (or FINAL) that appears in the RESET phrase, is (or would be) created. Thus, the RESET phrase serves the purpose of creating a total for the specified hierarchical level.

Example 8-12

```
01  ... TYPE CONTROL FOOTING DAY.
    02 ...
    .
    .
    .
    02  COLUMN 65 PIC $$$$9.99 SUM COST RESET ON FINAL.
01  ... TYPE CONTROL FOOTING FINAL.
    02 ...
    .
    .
    .
    02  COLUMN 45 PIC $$$$9.99 SUM SUM-DAY.
```

Because the SUM clause in the description of the control footing with the control item DAY contains the phrase RESET ON FINAL, the current value of the associated sum counter is printed every time the control footing of DAY is generated, without ever resetting the sum counter to zero. Only when the control footing for FINAL is generated will the sum counter be reset to zero. Therefore, each printed control footing for DAY shows the running cost figures from the first detail group of the report (1st day) through to the last detail group written before the current control heading.

A control data item that appears in a RESET phrase does not have to be associated with a control footing. A sum counter will be reset, as mentioned earlier, even when no control footing exists for a control item specified in a RESET entry.

11. Actions taken by the Report Writer Control System

When generating a control footing, the Report Writer Control System executes the following steps (schematically speaking, because steps may be omitted):

a) Adding hierarchically equivalent sums (crossfooting).

b) Execution of the USE BEFORE REPORTING procedures for the control footing (see "USE Statement").

c) PRINT SWITCH test.
If the value of the PRINT SWITCH special register is 1, step 4 is skipped, i.e. step 5 immediately follows step 3 after resetting the special register to zero. Otherwise, step 4 comes next.

d) Creation of the control footing (if printable).

e) Hierarchical incrementing (rolling-forward).

f) Any sum counters of the control footings whose SUM clauses do not contain RESET phrases are reset to zeros by implicit MOVE statements. The same applies to all those sum counters of the other control footings whose SUM clauses each contain one such RESET phrase which is referring to the control data item associated with the current (i.e. newly-created) control footing.

TYPE CLAUSE

Function

The TYPE clause indicates the type of the report group in whose description it appears; that is, it defines the functional characteristics of the report group, thereby also specifying the circumstances under which the Report Writer Control System will generate that report group.

Format

```
          ┌ ┌REPORT HEADING┐                                  ┐
          │ {              }                                  │
          │ └RH            ┘                                  │
          │ ┌PAGE HEADING┐                                    │
          │ {            }                                    │
          │ └PH          ┘                                    │
          │ ┌CONTROL HEADING┐ ┌data-name-1┐                   │
          │ {               } {           }                   │
          │ └CH             ┘ └FINAL      ┘                   │
          │ ┌DETAIL┐                                          │
TYPE IS  {  {      }                                          }
          │ └DE    ┘                                          │
          │ ┌CONTROL FOOTING┐ ┌data-name-2┐                   │
          │ {               } {           }                   │
          │ └CF             ┘ └FINAL      ┘                   │
          │ ┌PAGE FOOTING┐                                    │
          │ {            }                                    │
          │ └PF          ┘                                    │
          │ ┌REPORT FOOTING┐                                  │
          │ {              }                                  │
          └ └RF            ┘                                  ┘
```

Rules

1. RH is the abbreviation for REPORT HEADING.
   PH is the abbreviation for PAGE HEADING, etc.

2. FINAL, data-name-1 and data-name-2 must be defined in the CONTROL clause of the associated report description (RD) entry.

3. REPORT HEADING indicates the report group which is created only once per list as the first report group in that list. It is created automatically when the first GENERATE statement is executed.

4. PAGE HEADING indicates a report group which is created as the first group on every page. A page which is wholly reserved for the report heading or report footing is not assigned a page heading. If a report heading is present but does not appear on a page by itself, the page heading is generated as the second group on the first page of the report.

5. CONTROL HEADING indicates report groups which are written in series when executing the (chronologically) first GENERATE statement and upon every control break. Each control heading is associated (by FINAL or data-name-1) with one, and only one, hierarchical level, which determines the order in which the control headings are printed (see "CONTROL Clause").

6. DETAIL indicates those report groups which are written because they are supplied in a GENERATE statement. The name of a detail group must be unique throughout the report.

7. CONTROL FOOTING indicates those report groups which are written in series upon every control break and when the TERMINATE statement is executed. Each control footing is associated (by FINAL or data-name-2) with one, and only one, hierarchical level, which determines the order in which the control footings are printed (see "CONTROL Clause").

8. PAGE FOOTING indicates the report group which is generated as the last group of each report page. Any page that is wholly reserved for the report heading or report footing is not assigned a page footing. If the report footing does not appear on a page by itself, the page footing is generated as the last but one group on the last page of the report.

9. REPORT FOOTING indicates a report group that is produced only once, as the last group in the report. The report footing is the last group printed when the TERMINATE statement is executed.

Programming considerations

1. Rules for the report heading

   a) Only one report heading may be defined for each report.

   b) The report heading may be printed on a whole page by itself. The NEXT GROUP clause with the NEXT PAGE phrase specified may be used to make sure that the same page will contain no further group besides the report heading (see "NEXT GROUP Clause").

   c) The programmer may suppress printing the page heading if the page heading would be produced in addition to the report heading on the first page of the report. To do this, he should include a statement MOVE 1 TO PRINT-SWITCH in a USE BEFORE REPORTING declarative for the page heading. He must also make sure that this MOVE statement is not executed a second time (see "PRINT-SWITCH Special Register").

2. Rules for page heading and page footing

   a) Only one page heading and one page footing ma be defined for each report.

   b) Normally, the page heading appears as the first report group and the page footing appears as the last report group on every page of the report, with the following exceptions:

      - On the first page of the report, the page heading is preceded by the report heading in those cases where the report heading is not to appear on a page by itself.

- The page footing of the last page of the report is followed by the report footing in those cases where the report footing is not to appear on a page by itself.

- Neither a page heading nor a page footing may be printed on the page on which the report heading or the report footing is to appear by itself.

3. Rules for control headings and control footings

   a) Only one control heading and one control footing may be specified for each level of hierarchy.

   b) Control headings and control footings are printed at the following times:

      - When the first GENERATE statement for a report is executed, the Report Writer Control System prints the entire hierarchy of control heading report groups (in the order highest level first, lowest level last) before printing the detail group as the immediate product of the GENERATE statement.

      - If the Report Writer Control System detects a control break when executing one of the (chronologically) next GENERATE statements, it creates the appropriate series of control footings and control headings before the detail groups directly referenced by the current GENERATE statement (see "GENERATE Statement" and "CONTROL Clause").

      - When production of a report ends by execution of the TERMINATE statement, the Report Writer Control System generates all control footings in increasing order of hierarchy levels.

   c) FINAL, data-name-1 or data-name-2 specified in the TYPE clause of a control heading or control footing must appear in the CONTROL clause of the associated report description entry.

   d) A control heading or control footing associated with FINAL may only be generated once for each report, as the first body group of the report (execution of the first GENERATE statement) or as the last (execution of the TERMINATE statement).

4. Rules for detail groups

   The Report Writer Control System generates a detail report group only when it is specified in a GENERATE statement and when this GENERATE statement is executed. At least one detail report group must be defined for each report. This is true, even if the detail report group is not explicitly used for generating the report; that is, even if it is mainly a summary report without going into details (that is, detail report groups) (see "GENERATE Statement").

5. Rules for the report footing

   a) Only one report footing may be defined for each report, and it is printed as the last report group in the report.

   b) The report footing may appear on a page by itself, by specifying the first LINE clause with the NEXT PAGE phrase in the description of the report footing (see "LINE Clause").

6. Sequence of report groups within a report

   a) No report group of a report may be printed either before the report heading or following the report footing.

   b) The (schematic) sequence of the heading and footing groups for a report is this:

      Report heading (may only appear once)
      Page heading
      .
      .
      .
      Control heading
      Detail
      Control footing
      .
      .
      .
      Page footing
      Report footing (may only appear once)

   c) The control headings are always created in immediate succession in hierarchical order:

      Control heading with FINAL (supreme level, may only appear once)
      Control heading at next highest level
      .
      .
      .
      Control heading at lowest level

   d) The control footings are always created in immediate succession in hierarchical order:

      Control heading at lowest level
      Control heading at next lowest level
      .
      .
      .
      Control footing with FINAL (supreme level of hierarchy, may only appear once).

8

8.5 LANGUAGE ELEMENTS OF THE PROCEDURE DIVISION

GENERATE STATEMENT

Function

The GENERATE statement directs the Report Writer Control System to produce a portion of the report in accordance with the report description specified in the Report Section of the Data Division.

Format

$$\underline{\text{GENERATE}} \left\{ \begin{array}{l} \text{data-name} \\ \text{report-name} \end{array} \right\}$$

Rules

1. data-name must be defined in the Report Section of the Data Division, as the name of a detail report group (01-level entry).

2. The report-name must be defined as such (RD entry) in a report description entry of the Report Section in the Data Division.

3. As the result of a GENERATE statement referring to a detail report group, part of a detail is printed (see Rule 5).

4. As the result of a GENERATE statement referring to a report-name, part of a summary report is printed (see Rule 8).

5. As the result of a GENERATE statement referring to a detail report group, the Report Writer Control System generates part of the detail report group.The composition of this portion is indicated in Rules 6 and 7. In addition, various summations are generally executed (see "SUM Clause").

6. When executing the (chronologically) first GENERATE statement (relative to the execution of the related INITIATE statement), the following report groups (if defined) will be printed, provided this statement specifies a detail report group:

   a) report heading;

   b) page heading;

   c) all control headings from the highest to the lowest levels of hierarchy, and

   d) the detail report group specified in the GENERATE statement.

7. If a GENERATE statement specifying a detail report group is not executed as the (chronologically) first statement, the Report Writer Control System will first check whether a control break occurred. If so, the following report groups are written in the order stated:

   a) All control footings from the lowest level up to, and including, the level at which the control break occurred, and all control footings from the control break level down to the lowest level of hierarchy.

   b) The detail report group specified by the GENERATE statement.

   Step a) is not applicable unless there was a control break.

8. As a result of a GENERATE statement referring to a report, the Report Writer Control System takes the same actions, except for the creation of a detail report group, which is not applicable here. No additional actions beyond Rules 5, 6 and 7 will be taken.

9. When a page becomes full in the course of printing a report, the Report Writer Control System automatically generates a page advance, preceded by the page footing (if present) of the previous page and followed by the page heading of the new page.

Programming considerations

1. At the time of executing a GENERATE statement which specifies a detail report group, the following information must be available to the Report Writer Control System:

   a) All of the SOURCE clause information assigned to the detail report group and all other report groups to be created by the GENERATE statement.

   b) The numeric data of those addends the Report Writer Control System needs to accumulate the necessary sums.

2. Summary report printing is meaningful only for those reports for which control footings, too, are defined whose descriptions include SUM clauses.

3. Summary reporting should not be attempted for reports whose descriptions contain more than one detail report group because it does not allow relations to be established to the individual detail report groups.

4. The CBL-CTR special register (q.v.) is interrogated by the Report Writer Control System at the time of the (chronologically) first GENERATE statement. By using the MOVE statement to supply one of several defined values to this special register any time between the execution of the INITIATE statement and the (chronologically) first GENERATE statement, the programmer can select one or both of the following Report Writer options:

   a) Supply appropriate control data item values to the control footings, the page footing and the page heading.

   b) Condition execution of the NEXT GROUP clauses in the control footings (see "CBL-CTR Special Register").

INITIATE STATEMENT

Function

The INITIATE statement causes the Report Writer Control System to begin the processing of one or more reports.

Format

```
INITIATE report-name-1 [report-name-2]...
```

Rules

1. Each report-name supplied in the INITIATE statement must be defined as such (RD entry) by a report description in the Report Section of the Data Division.

2. The INITIATE statement indicates those reports (report-name-1, etc.) which the Report Writer Control System is to start generating.

3. The INITIATE statement instructs the Report Writer Control System to perform the following initialization functions for each named report:

   a) Set all sum counters to zero.

   b) Set LINE-COUNTER special register to zero.

   c) Set PAGE-COUNTER special register to one.

Programming considerations

1. The INITIATE statement by no means opens the file the named report is associated with. This report file, which is defined as a sequential output file, must therefore be OPENed as OUTPUT before execution of the INITIATE statement.

2. If an INITIATE statement is followed by a second (different) INITIATE statement specifying the same report as the first one, the latter INITIATE must not be executed for this report (although it may be for other reports), unless a TERMINATE statement specifying the same report was executed in between.

3. If an INITIATE and a TERMINATE statement were executed for a given report without an intervening GENERATE statement, the TERMINATE statement cancels the INITIATE statement without printing the report (or any portion thereof).

4. Following execution of the INITIATE statement, and prior to execution of the (chronologically) first GENERATE statement, the programmer may select certain optional functions of the Report Writer Control System by using the MOVE statement to supply the proper value to the CBT-CTR special register (q.v.).

TERMINATE STATEMENT

Function

The TERMINATE statement instructs the Report Writer Control System to complete the processing for the specified reports.

Format

```
TERMINATE report-name-1 [report-name-2]...
```

Rules

1. Each report-name specified in the TERMINATE statement must be defined as such (RD entry) in a report description of the Report Section in the Data Division.
2. The TERMINATE statement indicates those reports whose processing the Report Writer Control System is to complete.
3. For each report specified in the TERMINATE statement, the Report Writer Control System performs the following steps in the stated order:

   a) All control footings are created as if they were to be printed as the result of a control break at the supreme level of hierarchy (obviously, this implies creating the page footing and the page heading if a page advance is required, and the execution of summations).

   b) The page footing is generated.

   c) The report footing is generated.

   However, the above actions are not taken if no GENERATE statement was executed for the same report between the INITIATE and TERMINATE statements.

Programming considerations

1. Each TERMINATE statement for a report must be chronologically preceded by an INITIATE statement for that report.
2. Since the TERMINATE statement does not close the associated report file, it must be chronologically followed by a CLOSE statement. Each INITIATEd report in a particular file must be TERMINATEd before a CLOSE statement is executed for that file.

USE BEFORE REPORTING STATEMENT

Function

The USE BEFORE REPORTING statement introduces Procedure Division statements that are to be executed just before the specified report group is printed by the Report Writer Control System.

Format

USE BEFORE REPORTING report-group-name.

Rules

1. The report-group-name must be defined as a name (data-name) in a 01-level entry of a REPORT section report group entry in the Data Division. Any report group type except the detail report group may be specified in the USE BEFORE PRINTING statement.

2. The report-group-name identifies the report group for which the USE declaratives (USE BEFORE REPORTING procedures) following the USE BEFORE REPORTING statement are to be carried out.

3. The USE BEFORE REPORTING statement itself is never executed, but defines the calling conditions for the execution of the subsequently declared USE procedures.

4. The USE procedures declared for a report group are executed immediately before that report is created.

5. No more than 39 USE BEFORE REPORTING statements may appear in the Procedure Division.

Programming considerations

1. The name of a given report group may be specified in one declarative section only.

2. The INITIATE, GENERATE, and TERMINATE statements must not appear in a paragraph within any declarative section.

3. A USE BEFORE REPORTING procedure must not alter the value of any control field, nor any value of the subscripts used by the Report Writer Control System to access the control field.

4. The rules on references between a USE BEFORE REPORTING declarative and the remainder of the Procedure Division are the same as for other USE procedures.

5. The following are typical applications of USE BEFORE REPORTING procedures:

   a) Suppressing the printing of a report group:

      If the statement MOVE 1 TO PRINT-SWITCH is executed in a USE BEFORE REPORTING declarative, the Report Writer Control System will not print the report group the USE procedure was associated with. Since the Report Writer Control System always resets the PRINT-SWITCH special register to zero immediately after suppressing the report group (see PRINT-SWITCH), this register must be set to 1 again each time that printing is to be suppressed; with report groups that are written automatically, this can only be done by means of USE BEFORE REPORTING declaratives.

   b) Modifying the contents of a data item specified in a SOURCE clause:

      In editing a line with a printable item whose description includes a SOURCE clause, the contents of the item defined by the SOURCE clause is transferred to the printable item by means of an implicit MOVE statement. A USE BEFORE REPORTING procedure may be used to modify the contents of the sending field just prior to the execution of the implicit MOVE statement.

   c) Rounding sum counters:

      If the value of a sum counter is to be rounded before it is transferred to the associated print item for editing, by an implicit MOVE statement, this can only be achieved by means of a USE BEFORE REPORTING declarative for the control footing in which the sum counter was defined.

   d) If special actions depending on the level within the control hierarchy at which a control break has occurred are to be taken before writing a page heading, control heading, control footing, or page footing, this can only be effected via USE BEFORE REPORTING declaratives for this report group, owing to the automatic generation (see "SOURCE Clause" and "CBL-CTR Special Register").

8.6 SPECIAL REGISTERS OF THE REPORT WRITER

LINE-COUNTER SPECIAL REGISTER

LINE-COUNTER is a special register which is automatically defined for each report described in the REPORT section of the Data Division. The internal COBOL description of this special register is PICTURE S999 USAGE COMPUTATIONAL.

The Report Writer Control System always uses the LINE-COUNTER register to control the vertical spacing of the groups to be printed in a report. For this reason, the LINE-COUNTER register must not be changed by any Procedure Division statement.

As regards the PAGE LIMIT, NEXT GROUP, and LINE clauses, the special register LINE-COUNTER is automatically interrogated and incremented (or set to zero for page advance) by the Report Writer Control System. The INITIATE statement resets the LINE-COUNTER register to zero.

The LINE-COUNTER may be used both in the REPORT section and in the Procedure Division. As far as the REPORT section is concerned, LINE-COUNTER may only be specified in SOURCE clauses as indicated below:

```
SOURCE IS LINE-COUNTER [OF report-name]
```

As the Report Writer Control System always sets the LINE-COUNTER register to the current line (print line or NEXT GROUP positioning), the line number will be displayed on which the printable item associated with the above SOURCE clause is to be written. The LINE-COUNTER special register may be interrogated at any time in the Procedure Division; at the time of interrogation, it will contain the value assigned to it by the Report Writer Control System as a result of the NEXT GROUP clause of the last report group generated before the interrogation (last print line of the report group if no NEXT GROUP clause was specified).

If two or more reports are described in the Report Section, each explicit reference to a LINE-COUNTER must be qualified by the report-name.

PAGE-COUNTER SPECIAL REGISTER

PAGE-COUNTER is a special register that is defined automatically for each report described in the REPORT section of the Data Division. The internal COBOL description of this special register is
PICTURE S9(7) USAGE COMPUTATIONAL-3.

When the INITIATE statement is executed for a report, the Report Writer Control System increments the PAGE-COUNTER special register to 1 by means of an internal MOVE statement. As soon as the Report Writer Control System issues a page advance - after printing the page footing and before printing the page heading - it increments the current contents of the special register PAGE-COUNTER by 1.

The PAGE-COUNTER register is freely accessible. That is, its contents may be modified as well as interrogated. The most common use of PAGE-COUNTER is for printing page numbers in page headings or page footings. For this purpose, the PAGE-COUNTER register is assigned to a printable item of the associated report, by the following SOURCE clause:

```
SOURCE IS PAGE-COUNTER [OF report-name]
```

Program references to PAGE-COUNTER must be qualified by the report name if the Report Section describes more than one report.

PRINT-SWITCH SPECIAL REGISTER

PRINT-SWITCH is a special register that is defined automatically for a program whose Data Division includes the REPORT section. The internal COBOL notation of this register is PICTURE S9 USAGE COMPUTATIONAL-3. Only one such special register is defined for each source program.

The purpose of PRINT-SWITCH is to enable the program to suppress the printing of a report group. This is achieved by including the statement MOVE 1 TO PRINT-SWITCH within a USE BEFORE REPORTING procedure for the appropriate report group. The Report Writer Control System checks the contents of PRINT-SWITCH immediately after each execution of a USE BEFORE REPORTING declarative associated with a report group, and suppresses printing if PRINT-SWITCH contains a 1. After evaluating PRINT-SWITCH, the Report Writer Control System restores its value to 0, thereby preventing any inadvertent suppression of other report groups.

Report group suppression by the Report Writer Control System has the following effects:

a) No page spacing as specified by the LINE clauses of the report group.

b) No editing of print lines.

c) No page spacing as specified by any NEXT GROUP clause of the report group.

The suppression of a report group owing to the PRINT-SWITCH special register implies, in consequence of the above items 1 and 3, that the LINE-COUNTER special register is not changed.

PRINT-SWITCH should be set only within USE BEFORE REPORTING declaratives. If it is set anywhere else in the Procedure Division, it is generally impossible to predict which report group might be suppressed.

CBL-CTR SPECIAL REGISTER

CBL-CTR (Control Break Level Counter) is a special register that is defined automatically for each report described in the REPORT section of the Data Division. The internal COBOL notation of this special register is PICTURE S999 USAGE COMPUTATIONAL.

The Report Writer Control System and the object program use this special register to pass information to each other concerning control breaks and the action to be taken when control breaks occur.

CBL-CTR can be used in connection with two different functions. Either, neither, or both of these functions may be used for a given report. For the sake of simplicity, the two functions are known as "function 1" and "function 2". The user program notifies the Report Writer Control System which function it requires for a report. By a MOVE statement, the CBL-CTR register is assigned a value that symbolizes the requested function. Value and function interact as shown below:

Function Requested	MOVE Statement
Function 1	MOVE 1 TO CBL-CTR.
Function 2	MOVE 2 TO CBL-CTR.
Function 1 and 2	MOVE 3 TO CBL-CTR.

The corresponding MOVE statement must be executed after the INITIATE statement for the report but before the (chronologically) first GENERATE statement. This is the only time that the programmer is permitted to change the value of the CBL-CTR special register.

If two or more reports are defined in the REPORT section, each reference to CBL-CTR must be qualified by the report-name.

1. Function 1 of CBL-CTR

This function is invoked by a MOVE statement which places 1 (or 3) in the CBL-CTR special register.

This function consists of two parts; the respective actions are covered in the following rules.

a) Restoration of the previous values of the control fields in control footings.

Part 1 of function 1 makes previous values of control fields available to control footing SOURCE clauses or control footing USE BEFORE REPORTING declaratives.

SOURCE clauses (or USE BEFORE REPORTING declaratives) in control footings referring to control data items produce a problem. At the time the control footings are printed, some or all of the specified control fields have changed values. However, since the control footings printed because of a control break obviously belong to the precontrol break values. It is often desirable that the previous values (i.e. prior to the control break) should be printed. When function 1 of CBL-CTR is requested, these previous values of the control fields will be obtained by SOURCE clauses (or USE BEFORE REPORTING declaratives) of control footings.

For example, assume that the item MONTH-NAME is defined as a control field for a report and that the control footing MONTH-FOOTING is defined as follows:

```
01 MONTH-FOOTING TYPE CONTROL FOOTING
                      MONTH-NAME LINE PLUS 1.
   02 COLUMN 10 PICX(21) VALUE "***** END OF DATA FOR".
   02 COLUMN 33 PICX(9)  SOURCE MONTH-NAME.
```

In this case, the programmer wants the following control footing to be printed after all of the JANUARY data has been printed in the detail lines of the report:

```
***** END OF DATA FOR JANUARY.
```

Since the above control footing was printed because the item MONTH-NAME changed from JANUARY to FEBRUARY, FEBRUARY (the current contents) would be printed rather than JANUARY. By requesting function 1, the programmer can cause the prior value (JANUARY) to be printed instead of FEBRUARY.

b) Indicating the Control Break Level in CBL-CTR

Part 2 of function 1 causes the Report Writer Control System to place, in CBL-CTR, a value indicating the control break level in the hierarchy. This is done before control is passed to a USE BEFORE REPORTING procedure for execution. This kind of procedure begins with:

```
Section-name SECTION. USE BEFORE REPORTING report-name.
```

Indicating the level of the current control break in the hierarchy is accomplished by numbering the control fields consecutively, starting from the highest level in the hierarchy, not including FINAL. Thus, the first (leftmost) data-name of the CONTROL clause is numbered 1, the next is numbered 2, and so on.

For instance, assume that a report description entry contains this CONTROL clause:

```
CONTROLS ARE FINAL STATE COUNTY CITY
```

In this case, STATE is assigned number 1, COUNTY number 2, and CITY number 3.

The meanings of the CBL-CTR values in the USE procedures declared for the page heading and control headings are listed below in Table 9-4.

If, in the above example, the value of control field COUNTY changes, the value 2 will be placed in the CBL-CTR register by the Report Writer Control System (provided that the value of CITY has not changed at the same time also).

Value	Meaning
0	Indicates that no GENERATE statement has been executed or that the very first GENERATE statement is being executed.
1-254	Indicates that the control field with the corresponding number has just changed values, and that actions currently taking place were caused by this control break.
255	Indicates that no control break has occurred (cannot be encountered with control heading USE procedures).

Table 8-4 CBL-CTR values in USE procedures for page headings and control headings

The meanings of CBL-CTR values in USE declaratives for the page footing and control footings are listed below in Table 9-5.

Value	Meaning
0	Indicates the final stage of processing, i.e. the TERMINATE statement is being executed.
1-254	Indicates that the control field with the corresponding number has just changed values, and that actions currently taking place were caused by this control break.
255	Indicates that no control break has occurred (cannot be encountered with control footing USE procedures).

Table 8-5 CBL-CTR values in USE procedures for page footings and control listings

8

2. Function 2 of CBL-CTR

Function 2 of CBL-CTR is invoked by moving a value 2 (or 3) to CBL-CTR. This causes conditional execution of the NEXT GROUP clauses in the control footings.

Normally, the NEXT GROUP clause is executed immediately after the creation of the report group in whose description it is supplied (vertical spacing). If a control break causes several control footings to be printed in succession, the NEXT GROUP clauses of these control footings lead to inadvertent blank lines. Such blank lines are really intended for spacing from a control footing to a control heading or a detail report group but not to another control footing.

When function 2 of CBL-CTR is requested, the Report Writer Control System makes sure that only the last NEXT GROUP clause is executed, i.e. the NEXT GROUP clause associated with the last control footing generated in the series of control footings printed in succession as a result of a control break. Existing NEXT GROUP clauses of the other control footings of a closed sequence of control footings will be ignored. This is true even if the last control footing of the sequence has no NEXT GROUP clause.

9 SEGMENTATION FEATURE

9.1 GENERAL DESCRIPTION

The Segmentation Feature allows the programmer, at compile time, to specify object program overlay requirements. Only the Procedure Division may be segmented. The Procedure and Environment Divisions are therefore provided to define the segmentation requirements for the object program.

9.2 ORGANIZATION

Although it is not mandatory, the Procedure Division for a source program is usually written as several consecutive sections, each of which is composed of a series of statements which, taken together, perform a particular function. When segmentation is used, the entire Procedure Division must be in sections. In addition, each section must be classified as belonging either to the fixed portion of the object program or to one of the independent segments of the program. On the other hand, segmentation in no way affects the need for qualification of procedure-names to ensure uniqueness.

9.3 FIXED PORTION OF THE OBJECT PROGRAM

The fixed portion of the object program is defined as that portion that is logically treated as if it were always resident in computer storage. This portion of the program is composed of two types of computer storage segments, permanent segments and overlayable fixed segments.

a) A permanent segment is a segment in the fixed portion that cannot be overlaid by another part of the program.

b) An overlayable fixed segment is a segment in the fixed portion which can overlay, or be overlaid by, either another overlayable fixed segment or an independent segment in order to optimize internal storage utilization. From the execution point of view, however, it is considered to be resident in internal storage. That, is, such a segment, if called, is always made available in the state in which it was last used.

Depending on the availability of internal storage, the number of the permanent segments in the fixed portion may be varied through the use of the SEGMENT-LIMIT clause, which is to be discussed later in this chapter.

9.4 INDEPENDENT SEGMENTS

An independent segment is defined as that part of the object program that can overlay other segments and can be overlaid by an overlayable fixed segment or by another independent segment. If procedures contained within an independent segment are referenced by a PERFORM or GO TO statement from outside that independent segment, the program is always provided with an independent segment which is in its initial state.

9.5 GENERAL RULES FOR SEGMENTATION

1. The logical sequence of the program is the same as the physical sequence except for specific transfers of control. If the sections that belong to a given segment, i.e. sections which have the same segment number, are scattered through the source program, they must be reordered by the compiler. However, the compiler will provide transfers of control to maintain the logic flow of the source program. The compiler will also insert instructions necessary to load and/or initialize a segment as necessary. Control may be transferred within a source program to any paragraph in a section; that is, it is not mandatory to transfer control to the beginning of a section.

2. Only the permanent segments remain in internal storage through-out program execution. Both the overlayable fixed segments and the independent segments may overlay one another.

3. The following criteria should be noted when classifying segments:

   Logical requirements:
   Sections that must be available for reference at all times, or which are referred to very frequently, are normally classified as belonging to one of the permanent segments; sections that are less frequently used are normally classified as belonging either to one of the overlayable fixed segments or to one of the independent segments, according to the logic requirements of program flow.

   Frequency of use:
   Generally, the more frequently a section is referred to, the lower should be its segment number; the less frequently it is referred to, the higher should be its segment number.

   Relationship to other sections:
   Sections that frequently communicate with one another should be given equal segment numbers. All sections with the same segment number constitute a single program.

4. When segmentation is used, the following rules apply to the ALTER, PERFORM, MERGE and SORT statements as well as the called program:

ALTER statement:

a) A GO TO statement in a section whose segment number is greater than or equal to 50 must not be referenced by an ALTER statement in a section with a different segment number.

b) A GO TO statement in a section whose segment number is less than 50 may be referenced by an ALTER statement in any section, even when the GO TO statement referenced by the ALTER statement belongs to a program segment which has not yet been called for execution.

PERFORM statement:

a) A PERFORM statement that appears in a section whose segment number is less than the segment number supplied in the SEGMENT-LIMIT clause can have within its range only the following:

- Sections each having a segment number less than 50.

- Sections wholly contained in a single segment whose segment number is greater than 49.

However, the compiler permits the PERFORM statement to reference, within its range, section having any section number.

b) A PERFORM statement appearing in a section whose segment number is equal to or greater than the priority number specified in the SEGMENT-LIMIT clause can have within its range only one of the following:

- Sections each having the same segment number as that of the section containing the PERFORM statement.

- Sections each having a segment number less than that specified in the SEGMENT-LIMIT clause.

However, the compiler permits the PERFORM statement to reference, within its range, sections having any section number.

SORT/MERGE statement:

a) When using a SORT or MERGE statement within a section which is not an independent segment, the input procedures or output procedures referenced by that SORT or MERGE statement:

- must be contained wholly within independent segments,

- or must be contained wholly within a single dependent segment.

b) When using a SORT or MERGE statement within an independent segment, the input procedures or output procedures referenced by that SORT or MERGE statement:

- must be wholly contained within independent segments, or

- must be wholly contained within the same independent segment as the SORT or MERGE statement.

These restrictions do not apply to the compiler discussed in this manual.

9.6.1 Language Elements of the Environment Division

SEGMENT-LIMIT CLAUSE

Function

The SEGMENT-LIMIT clause enables the user to vary the number of permanent and overlayable fixed segments in his program, while still retaining the logical properties of fixed portion segments (segment number 0 through 49) and keeping constant the total number of segments in the fixed portion.

Format

<u>SEGMENT-LIMIT</u> IS segment number

Rules

1. The SEGMENT-LIMIT clause is supplied in the OBJECT-COMPUTER paragraph. This is an optional clause.

2. segment number must be an unsigned integer that ranges in value from 1 through 50.

3. When the SEGMENT-LIMIT clause is specified, only those segments having segment numbers up to, but not including, the segment number designated in the SEGMENT-LIMIT clause are considered as permanent.

4. Those segments having segment numbers from the segment limit through 49 are considered to be overlayable fixed segments.

5. When the SEGMENT-LIMIT clause is omitted, all segments having segment numbers from 0 through 49 are considered as permanent segments of the object program.

Programming consideration

Ideally, all program segments having segment numbers ranging from 0 through 49 are treated as permanent segments. However, when insufficient storage is available to contain all permanent segments plus the largest overlayable segment, the number of permanent segments must be reduced. This may be done by subsequently including a SEGMENT-LIMIT clause with appropriate entries without otherwise changing the program.

<u>Example</u>

SEGMENT-LIMIT IS 40

This clause indicates that all segments having segment numbers from 0 through 39 are considered to be permanent segments of the object program. Segments having segment numbers from 40 through 49 are overlayable fixed segments.

9.6.2 Language Elements of the Procedure Division

SEGMENT NUMBER

Function

Section classification is achieved by a system of segment numbers. The segment number is included in the section heading.

Format

---

section-name <u>SECTION</u> [segment number].

Rules

1. Section-name identifies the section.

2. Segment number indicates whether the chapter belongs to a permanent, overlayable fixed, or independent segment.

3. segment number must be an unsigned integer ranging in value from 0 through 99.

4. All sections that have the same segment number constitute a program segment with that number.

5. Segments with segment numbers 0 through 49 belong to the fixed portion of the object program. The SEGMENT-LIMIT clause indicates which of these segments are treated as permanent and which as overlayable fixed segments.

6. Segments with segment numbers 50 through 99 are independent segments.

7. If the segment number is omitted from the section heading the segment number is assumed to be 0.

8. Sections in the DECLARATIVES subdivision of the Procedure Division must not contain segment numbers in their section headings. They are treated as permanent segments whose segment number is 0.

Programming consideration

When a procedure-name in an independent segment is referred to by a PERFORM or GO TO statement contained in a segment with a different segment number, the segment referred to is made available in its initial state for each execution of the PERFORM or GO TO statement. If the PERFORM statement is repeated, the initial state is restored only for the first execution of the PERFORM statement.

10 SORT-MERGE FEATURE

10.1 SORTING AND MERGING OF FILES

10.1.1 Sort Processing

The user has the capability of sorting a file on the basis of a series of keys. These sort keys are specified by the user and are present in each record of the file. The records may be sorted so that all of their keys are in ascending or in descending order, or so that some of their keys are ascending and others are descending.

Records are sorted on a file called a sort-file. This file is defined in the Data Division in a sort-file description (SD) entry, and in the record description associated with the SD. The SORT statement in the Procedure Division initiates the sort operation.

Sort processing consists of:

a) Releasing all records to the sort-file.
b) Sorting the records on the sort-file.
c) Returning all records from the sort-file.

The user may either provide an input procedure to process records and transfer them to the sort-file, or he may specify an input file containing the records to be sorted and allow the SORT statement to transfer these records to the sort-file. Accordingly, he may either provide an output procedure to retrieve records from the sort-file and process them further, or he may specify an output file and allow the SORT statement to output the sorted records to this file.

Several sort operations may be specified in a single program.

10.1.2 Merge Processing

The user has the capability of merging from two to 16 sorted input files with the same record format and transferring them to an output file.

Merging takes place in a file called a sort-file. This file is defined in the Data Division within a sort-file description (SD) entry, and in the record description associated with the SD. Merge processing is initiated by means of the MERGE statement in the Procedure Division.

Merge processing consists of:

a) Releasing the records of all input files to the sort-file.
b) Merging the records in the sort-file.
c) Returning all records from the sort-file.

The MERGE statement releases the records of all specified input files to the sort-file. The user can likewise specify an output file into which the MERGE statement returns the merged records, In connection with returning, it is also possible to specify an output procedure. This procedure accepts records from the sort-file and continues processing them.

Several merge operations may be specified in a single program.

10.1.3 Sort and Merge without Input/Output Procedures

If no input or output procedures are specified, the compiler will generate procedures to take charge of record input/output. In this case the user must describe the following files:

a) one or more input files containing the records to be sorted or merged,

b) one sort-file and one or more output files to which the records will be returned.

When referenced, a SORT or MERGE statement performs the following functions:

1. Opens the input and sort-files.

2. Releases all the records in the input file to the sort-file.

3. Sorts or merges the records on the sort-file.

4. Closes input files, opens output files.

5. Returns the records from the sort-file to the output file.

6. Closes the sort, and output files.

10.1.4 Sort with Input/Output Procedures of the User

If input and output procedures are specified, the following actions are taken:

1. When a SORT statement is executed, it transfers control to the input procedure.
2. The input procedure performs the following functions:
   a) Processes a record (for example, reads a record from a file or creates a new record).
   b) Releases the record to the sort-file.
   c) Repeats steps a) and b) until all records have been released.
3. The SORT statement sorts the records on the sort-file.
4. Control is then passed to the output procedure.
5. The output procedure performs the following functions:
   a) Accepts a record from the sort-file.
   b) Processes the record (for example, writes the record to an output file).
   c) Repeats steps a) and b) until all records have been returned and processed.
6. Control is returned to the statement following the SORT statement.

If options are mixed (e.g. if only an input procedure is specified), appropriate variants of the above procedures are performed.

10.1.5 Sort and Merge with Output Procedures but without Input Procedure of the User

1. Opens input, sort, and output files.
2. Releases all the records from the input files to the sort-file.
3. Sorts the records in the sort-file.
4. Control is then returned to the output procedure.
5. The output procedure performs the following functions:
   a) Accepts a record from the sort-file.
   b) Processes the record (for example, writes the record to an output file).
   c) Repeats steps a) and b) until all records have been returned and processed.
6. Control is transferred to the statement following the SORT/MERGE statement.

10.2 LANGUAGE ELEMENTS

To use the sort feature, the user must provide additional information in the Environment, Data, and Procedure Divisions of the source program. This information is summarized in the following table and is described in detail in the remainder of this section.

Source program division	Contents and meaning
Environment Division	A SELECT clause in the FILE-CONTROL paragraph for the sort-file (sort-file-name).
	SELECT clauses in the FILE-CONTROL paragraph for all files used as input and output procedures for the sort-file (section-name-1, section-name-2, section-name-3, section-name-4 for SORT; section-name-1, section-name-2 for MERGE) and for files appearing in the USING or GIVING phrase of a SORT or MERGE statement (file-name-1,...).
	To permit restart of object programs containing the SORT statement, the RERUN clause may be used in the I-O-CONTROL paragraph.
	The SAME SORT AREA or the SAME SORT-MERGE AREA clause in the I-O-CONTROL paragraph is intended to optimize the allocation of internal storage to a sort-file.
Data Division	A sort-file description (SD) entry and associated record description entries for the sort-file; record descriptions must include sort-key fields.
	File description (FD) entries and associated record description entries for all files used in input/output procedures for the sort-file section-name-1, section-name-2, section-name-3, section-name-4 for SORT; section-name-1, section-name-2 for MERGE) and for files appearing in the USING or GIVING phrase of a SORT or MERGE statement (file-name-1,...).
	Data description entries for the above files.

<table>
<tr><th>Source program division</th><th colspan="2">Contents and meaning</th></tr>
<tr><td rowspan="7">Procedure Division</td><td colspan="2">A SORT or MERGE statement for the sort-file specifying the following information:

. Sort-key names
. Whether the sort is to be in ascending or descending order of key.
. Input/output information specified by the following options:</td></tr>
<tr><td>Specification</td><td>Meaning</td></tr>
<tr><td>INPUT PROCEDURE</td><td>Records will be released to the sort-file by an input procedure.</td></tr>
<tr><td>USING file-name-1,...</td><td>Specifies files from which the SORT statement will accept records to be released to the sort-file.</td></tr>
<tr><td>OUTPUT PROCEDURE

GIVING file-name-3,...</td><td>Records will be returned from the sort-file by an output procedure.
Specifies a file to which the SORT or MERGE statement will return sorted records.</td></tr>
<tr><td colspan="2">If input and/or output procedures are specified in the SORT or MERGE statement, these procedures must be included in the Procedure Division. An input procedure must include a RELEASE statement, to transfer records to the sort-file. An output procedure must include a RETURN statement, to accept records from the sort-file.</td></tr>
<tr><td colspan="2">Special sort-registers can be referenced in the Procedure Division (see "Special registers for SORT").</td></tr>
</table>

10.2.1 Language Elements of the Environment Division

RERUN CLAUSE

Function

The RERUN clause makes it possible for programs containing a SORT or MERGE statement to be restarted.

Format

```
          ┌file-name-1        ┐
RERUN ON  ┤                   ├ EVERY SORT
          └implementor-name   ┘

     [OF-file-name-3[...]]
```

Rules

file-name-3 must be defined in an FD entry.

Programming considerations

1. When the OF phrase is not specified, the checkpoint is written prior to each sort operation.

2. When file-name-3 is specified, checkpoints are written for this specific sort operation only.

SAME AREA CLAUSE

Function

The SAME AREA clause indicates that two or more files are to share a specified internal storage area during program execution.

Format

```
     ┌RECORD    ┐
SAME ┤SORT      ├ AREA FOR file-name-1 [file-name-2]...
     └SORT-MERGE┘
```

Rules

1. In the SAME AREA clause, SORT and SORT-MERGE are equivalent.

2. When the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file-names must refer to a sort or merge file. Files which are not sort or merge files may also be specified in the clause.

3. The three allowable formats of the SAME AREA clause (i.e. SAME RECORD AREA, SAME SORT AREA, SAME SORT-MERGE AREA) are discussed in detail below. More than one SAME AREA clause may appear in a given program; however:

   a) A given file-name must not occur in more than one SAME RECORD AREA clause.

   b) A file-name representing a sort-file may not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.

   c) If a file-name which does not represent a sort-file appears in a SAME AREA clause and in one (or more) SAME SORT AREA or SAME SORT-MERGE AREA clause(s), then all files specified in the SAME AREA clause must also be supplied in the SAME SORT AREA or SAME SORT-MERGE AREA clause.

4. The files that appear in the SAME SORT AREA, SAME SORT-MERGE AREA or SAME RECORD AREA clause need not all have the same organization or access mode.

Programming consideration

The SAME RECORD AREA clause indicates that two or more files should share the storage area in which the current logical record is being processed. All of the files can be open at the same time. A logical data record in the "same record area" is considered as a logical data record of each opened output file whose file-name occurs in the SAME RECORD AREA clause; it is also considered as a logical data record of the last read input file whose file-name occurs in the SAME RECORD AREA clause. This is equivalent to an implicit redefinition of the internal storage area, where the data records are aligned on the leftmost character position.

SELECT CLAUSE

Function

The FILE-CONTROL paragraph is specified only once in a COBOL program; within this paragraph, one SELECT clause must be provided for each file referenced in the program.

Files used in input and output procedures, as well as files occurring in the USING or GIVING phrases of a SORT statement, are specified in a SELECT clause, as described in the Environment Division.

A file occurring in the GIVING phrase of a SORT or MERGE statement may be described as specified in format 1 (see chapter 3), whereas format 2 must be used for a sort-file.

Format 2

```
SELECT sort-file-name

              ⎡[integer-1] SORT-TAPES                                       ⎤
              ⎢                                                             ⎥
              ⎢[integer-2] DISC                                             ⎥
     ASSIGN TO⎨                                                             ⎬
              ⎢[integer-3] MERGE                                            ⎥
              ⎢                                                             ⎥
              ⎣[integer-3] implementor-name-1 [implementor-name-2]...       ⎦
```

Rules

1. sort-file-name indicates a sort-file.
2. For the format of implementor-name-1, ..., see "ASSIGN Clause".
3. All other entries are treated as comment.

Programming considerations

1. Format 2 is applicable only to a SELECT clause referring to a sort-file description entry. No further clauses other than the ASSIGN clause are permitted.
2. Each sort-file described in the Data Division must be designated once, and only once, as a file-name in the FILE-CONTROL paragraph.
3. Each sort-file specified in the FILE-CONTROL paragraph must have a sort-file description entry in the Data Division.

4. The following file-names must not be used in SELECT clauses within a program that uses SORT and runs under BS2000:

```
MERGExx(xx = 01,...,99)
SORTIN
SORTINxx(xx = 01,...,99)
SORTOUT
SORTWK
SORTWKx(x = 1,...,9)
SORTWKxx(xx = 01,...,99)
SORTCKPT
```

10

10.2.2 Language Elements of the Data Division

SORT-FILE DESCRIPTION

Function

A Sort-file Description (SD) entry provides information concerning the physical structure, identification, and size of the records on a sort-file.

Format

```
SD  sort-file-name
   [RECORDING MODE IS mode]

   [      {RECORDS ARE} {OMITTED } ]
   [LABEL {           } {        } ]
   [      {RECORD IS  } {STANDARD} ]

   [     {RECORD IS  }                              ]
   [DATA {           } data-name-1 [data-name-2]... ]
   [     {RECORDS ARE}                              ]

   [RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS]
```

Rules

1. sort-file-name indicates the sort-file.
2. data-name-1, data-name-2,... refer to records described in the record description entries associated with this sort-file description (SD) entry.

Programming considerations

1. The level indicator SD identifies the beginning of the sort-file description entry and must precede the file-name.
2. The clauses following the name of the file are optional and may appear in any order.
3. One or more record description entries for data-name-1,... must follow the sort-file description entry.
4. The recording mode for a sort-file must be specified as either F or V in the RECORDING MODE clause.
5. The FILE SECTION must contain a sort-file description entry for each sort-file, i.e. for each file that is supplied as the first operand within a SORT or MERGE statement.
6. The RECORDING MODE clause specifies the organization format of data on external devices.

7. The LABEL RECORDS clause is optional; however, if it is not specified LABEL RECORDS ARE OMITTED is assumed. This means that existing labels will be overwritten.

8. When the LABEL RECORDS ARE STANDARD clause is specified, the work tapes must have standard labels for sorting purposes; however, label die handling is not performed. In this case, the labels are retained intact.

9. The DATA RECORDS clause specifies the names of the data records on the file to be sorted.

10. Occurrence of more than one data-name indicates that this file contains two or more types of data records. These records may differ in length, format, etc. They may be listed in any order.

11. Conceptually, all data records of a file share the same area. This is in no way affected by the presence of more than one type of data record on that file.

12. The RECORD CONTAINS clause specifies the size of the data records in the file to be sorted.

13. The RECORD CONTAINS clause is used only as documentation. Record length is the sum of the lengths of the elementary data items.

14. The RECORDING MODE, DATA RECORDS and RECORD CONTAINS clauses are optional, as the compiler can determine the modes, names and sizes of the records from the associated record descriptions (complete details on these clauses are given in the Data Division).

15. If any of the record descriptions associated with the sort-file description contains an OCCURS clause with the DEPENDING ON phrase, variable-length data records are assumed (for further information on assumptions made by the compiler when the RECORDING MODE clause is omitted, see "RECORDING MODE Clause").

16. Sort-file-names may only be used in the SORT, MERGE and RETURN statements..

10

10.2.3 Language Elements of the Procedure Division

MERGE STATEMENT

Function

The MERGE statement creates a sort-file into which records are accepted from two or more similarly sorted input files. It merges the records in the sort-file on the basis of a series of given data items (sort-keys) and, once this merge operation is finished, makes each record from the sort-file available to an output procedure or to output files.

Format

```
MERGE sort-file-name

        ON {DESCENDING} KEY data-name-1 [data-name-2]...
           {ASCENDING }

      [ ON {DESCENDING} KEY data-name-3 [data-name-4]... ]...
      [    {ASCENDING }                                  ]

     [COLLATING SEQUENCE IS alphabet-name]

     USING file-name-1, file-name-2 [,file-name-3]...

     {OUTPUT PROCEDURE IS section-name-1 [{THRU   } section-name-2]}
     {                                   [{THROUGH}               ]}
     {GIVING file-name-4 [,file-name-5]...                         }
```

Rules

1. sort-file-name must be defined in a sort-file description (SD) entry in the Data Division.

2. sort-file-name must correspond to the sort-file-name defined in the SELECT clause (format 2).

3. data-name-1,..., data-name-3,... are sort-keys. A sort-key is that part of a data record which is used as a basis for sorting. Sort keys must be defined in a record description belonging to an SD description entry. They are subject to the following rules:

   a) The data items must not be of variable length.

   b) The data-names describing the keys may be qualified (see "Qualification").

c) When two or more data record descriptions are supplied, the keys need only be described in one of these descriptions. If a key is defined in more than one record description, the descriptions of that key must be identical, and must ensure that the key appears in the same position within each record.

d) A sort-key must not be defined with an OCCURS clause and must not be subordinate to a table item defined with an OCCURS clause.

e) A maximum of 12 keys may be specified for any file.

f) The total length of all keys must not exceed 256 bytes.

g) A key may appear anywhere in a data record, but the keys of all records to be sorted in a given sort run must be equidistant from the beginning of the record. this distance must not exceed 4,092 bytes.

h) Keys are always listed in the SORT statement from left to right in the order of their decreasing significance, regardless of whether they are ascending of descending. Hence, data-name-1 would be the principal sort-key and data-name-2 the subsidiary key.

j) A sort-key, when expressed as a packed decimal, may have no more than 16 digits.

4. section-name-1 identifies the first or only section in the output procedure. Section-name-2 identifies the last section in the output procedure. It is required only if the output procedure consists of more than one section.

5. file-name-1, file-name-2, file-name-3,..., file-name-4, file-name-5,... must be defined in a file description (FD) entry in the Data Division.

6. file-name-1, file-name-2,..., file-name-3, file-name-4,... must be

   a) sequential files.

   b) The size of each logical data record described for these files must be equal to the size of each data record described for the sort-file. When the data descriptions of the elements which make up these records are not identical, the user must describe these records in such a way that they occupy the same number of character positions in main memory.

7. At least one ASCENDING/DESCENDING phrase must be specified in a MERGE

   statement.

8. The MERGE statement may be written anywhere in the program except

   a) in the declaratives area,

   b) in an output procedure belonging to a MERGE statement.

Programming considerations

1. The collating sequence is set by means of the ASCENDING/DESCENDING option:

   a) When ASCENDING is specified, sorting proceeds from the lowest to the highest value of the sort-key, i.e. in ascending order.

   b) When DESCENDING is specified, sorting proceeds from the highest to the lowest value of the sort-key, i.e. in descending order.

   The collating sequence is governed by the same rules as apply to the comparison of operands in "Relation Conditions".

2. When the program is executed, the collating sequence for the comparison of nonnumeric sort items is set as follows:

   a) If COLLATING SEQUENCE has been specified in the MERGE statement, this entry is used as a sort criterion.

   b) If COLLATING SEQUENCE was not specified in the MERGE statement, the program-specific collating sequence will be used (see "OBJECT-COMPUTER Paragraph").

3. file-name-1, file-name-2, file-name-3,... in the USING phrase is the name of the input file to be sorted. All records contained therein will automatically be transferred to the file indicated by sort-file-name. Thus, file-name-1, file-name-2, file-name-3,... may not be opened while the MERGE statement is executing.

   The MERGE statement opens, processes and closes this input file automatically.

4. If the programmer specifies error handling and/or label processing declaratives for file-name-1, file-name-2, file-name-3,..., the MERGE statement will make the necessary linkage to the appropriate Declaratives Section.

5. OUTPUT PROCEDURE indicates that the Procedure Division contains an output procedure to process records after they have been merged. If OUTPUT PROCEDURE is specified, control passes to it after the sort-file has been processed by the MERGE statement. During RETURN statement processing, the output procedure accepts the records from the sort-file. The user may not leave the output procedure. The compiler inserts a return mechanism at the end of the last section of the output procedure. When control passes to the last statement in the output procedure, the return mechanism provides for termination of the sort, and then passes control to the statement following the MERGE statement.

   The following rules apply to the output procedure, which is a self-contained section within the Procedure Division:

   a) It must consist of one or more sections that are written consecutively and do not form part of an input procedure for a SORT statement.

   b) It must contain at least one RETURN statement, to make merged records available for processing.

   c) It may run only in conjunction with an associated MERGE statement.

   d) It may include any procedures needed to select, modify, or copy the records as they are returned.

Restrictions:

- The output procedure must not contain any SORT or MERGE statements.

- It is forbidden to branch from within an output procedure to procedure-names outside the procedure.

  However, the compiler permits the ALTER, GO TO and PERFORM statements in the output procedure to refer to procedure-names outside the output procedure. The programmer must make sure that a transfer of this type is followed by a return to the output procedure in order to effect a proper exit from this procedure (i.e. to processing its last statement).

- It is forbidden to branch from points outside an output procedure to procedure-names within that procedure.

  However, the compiler permits this provided the branch does not involve a RETURN statement or the end of the output procedure.

6. When the OUTPUT PROCEDURE phrase is used, control is passed from the specified procedure as though a format-1 PERFORM statement is executing. That is, all sections constituting the procedure are executed once, and execution of the procedure is terminated after its last statement has been processed. Thus, any procedure may be terminated by using an EXIT statement.

7. If MERGE statements are supplied in segmented programs, the following restrictions apply:

   a) If a MERGE statement appears in a section which is outside any independent segment, then all input or output procedures referred to by that MERGE statement must be:

      - either wholly contained within one fixed segment,

      - or wholly contained in a single independent segment.

   b) If a MERGE statement appears in an independent segment, then all input or output procedures referred to by that MERGE statement must be:

      - either wholly contained within one fixed segment,

      - or wholly contained in the same independent segment as the MERGE statement.

   These restrictions do not apply to the compilers discussed in this manual.

8. The GIVING phrase is specified when there is no output procedure. File-name-4, file-name-5, ... is the name of the output file. All the records from the file indicated by sort-file-name are automatically released to this output file. Thus, file-name-4, file-name-5, ... may not be opened while the MERGE statement is executing. The MERGE statement opens, processes and closes this output file automatically.

9. If the programmer specifies error handling and/or label processing declaratives for file-name-4, file-name-5, ..., the MERGE statement will make the necessary linkage to the appropriate Declaratives Section.

10. The terminal routine for all files is processed as if a CLOSE statement, without optional operands, is executed for each file involved.

RELEASE STATEMENT

Function

The RELEASE statement transfers data records from the input file to the sort-file. It can only be used during a sort operation.

Format

RELEASE sort-record-name [FROM identifier]

Rules

1. sort-record-name must be the name of a logical record in the associated sort-file description.

2. sort-record-name and identifier must not refer to the same internal storage area.

Programming considerations

1. Execution of a RELEASE statement has the effect that the data record specified by sort-record-name is transferred to the sort-file, to be further processed by the SORT routine of the system.

2. The FROM phrase makes the RELEASE statement equivalent to a MOVE statement followed by a RELEASE statement.

   When this phrase is written, data is moved from the area specified by identifier to the area identified by sort-record-name, and is then released to the sort-file. The move takes place according to the rules which govern the MOVE statement without the CORRESPONDING phrase.

3. A RELEASE statement may be used only within an input procedure in connection with a SORT statement.

4. After a RELEASE statement is executed, the logical data record is no longer available in the record area, unless the associated sort-file, appears in a SAME RECORD AREA clause. The logical data record is also at program execution time, available as a data record of other files specified in the SAME RECORD AREA clause as the associated sort-file; and it is available to the file associated with sort-record-name. When control is transferred to the input procedure, that file consists of all those data records which were passed by means of the execution of RELEASE statements.

5. After a RELEASE statement with the FROM phrase is executed, the data record is still available in "identifier".

10

RETURN STATEMENT

Function

The RETURN statement obtains individual records in sorted order from the sort-file.

Format

```
RETURN sort-file-name RECORD [INTO identifier]
       AT END imperative-statement.
```

Rules

1. sort-file-name must be defined in a sort-file description entry.

2. The storage area associated with identifier and the record area associated with sort-file-name must not be the same internal area.

Programming considerations

1. Execution of a RETURN statement has the effect that the next record is made available, according to the order specified by the keys of the SORT/MERGE statement. Then it can be processed in the data record areas of the sort-file.

2. If more than one data record description is supplied for the logical record of a file, these records will automatically share the same storage area; this corresponds to an implicit redefinition of the area. The contents of any data item which may be outside the area of the current data record will be unspecified after a RETURN statement is executed.

3. The INTO phrase, when specified, makes the RETURN statement equivalent to a RETURN statement followed by a MOVE statement.

4. When this phrase is written, records are returned from the sort-file and then moved to the area specified by identifier; identifier must not refer to a data item within the data record of that sort-file. The move is made according to the rules specified for a MOVE statement without CORRESPONDING phrase.

5. The MOVE statement is not carried out if an AT END condition occurs.

6. Any subscripting or indexing of identifier is evaluated after the record has been obtained and immediately before it is moved into the data area.

7. If the INTO phrase was used, the data will be available both in the input record area and in the data area specified by identifier.

8. Imperative-statement specifies an action to be taken when all of the sorted records have been obtained from the sort-file.

9. If no next logical record exists at the time a RETURN statement is executed, the AT END condition is true. The contents of the record areas associated with the file is unspecified if the AT END condition occurs. After execution of the imperative statement supplied in the AT END phrase, no further RETURN statement associated with the current output procedure can be executed.

10. A RETURN statement may be used only within the range of the output procedure associated with a SORT/MERGE statement for sort-file-name.

SORT STATEMENT

Function

The SORT statement is used to sort data records either created in an input procedure or contained in a file according to a series of given data items (sort-keys). The sorted records are released to an output procedure or entered in a file.

Format

```
SORT sort-file-name

          ⎧DESCENDING⎫
       ON ⎨          ⎬ KEY data-name-1 [,data-name-2]...
          ⎩ASCENDING ⎭

     ⎡    ⎧DESCENDING⎫                                ⎤
     ⎢ ON ⎨          ⎬ KEY data-name-3 [,data-name-4]...⎥...
     ⎣    ⎩ASCENDING ⎭                                ⎦

     [WITH DUPLICATES IN ORDER]

     [COLLATING SEQUENCE IS alphabet-name]

     ⎧                                    ⎡⎧THRU   ⎫                ⎤⎫
     ⎪ INPUT PROCEDURE IS section-name-1  ⎢⎨       ⎬ section-name-2 ⎥⎪
     ⎨                                    ⎣⎩THROUGH⎭                ⎦⎬
     ⎪                                                               ⎪
     ⎩ USING file-name-1 [,file-name-2]...                           ⎭

     ⎧                                    ⎡⎧THRU   ⎫                ⎤⎫
     ⎪ OUTPUT PROCEDURE IS section-name-3 ⎢⎨       ⎬ section-name-4 ⎥⎪
     ⎨                                    ⎣⎩THROUGH⎭                ⎦⎬
     ⎪                                                               ⎪
     ⎩ GIVING file-name-3 [,file-name-4]...                          ⎭
```

Rules

1. sort-file-name must be described in a sort-file description (SD) entry in the Data Division.

2. sort-file-name must correspond to the sort-file-name defined in the SELECT clause (format 2).

3. data-name-1,..., data-name-3,... are sort-keys. A sort-key is that part of a data record which is used as a basis for sorting. Sort-keys must be defined in a record description belonging to an SD entry. They are subject to the following rules:

   a) The data items must not be of variable length.

   b) The data-names describing the keys may be qualified (see "Qualification").

c) When two or more data record descriptions are supplied, the keys need only be described in one of these descriptions. If a key is defined in more than one record description, the descriptions of that key must be identical, and the descriptions must ensure that the key appears in the same position within each record.

d) A sort-key must not be defined with an OCCURS clause and must not be subordinate to an item defined with an OCCURS clause.

e) A maximum of 12 keys may be specified for any file.

f) The total length of all keys must not exceed 256 bytes.

g) A key may appear anywhere in a data record, but the keys of all records to be sorted in a given sort run must be equidistant from the beginning of the record. This distance must not exceed 4,092 bytes.

h) Keys for sorting purposes are always listed from left to right in order of significance, regardless of whether they are ascending or descending. Hence, data-name-1 is always the principal sort-key and data-name-2 the subsidiary key.

j) A sort-key, when expressed as a packed decimal, may be no more than 16 digits long.

4. section-name-1 identifies the first or only section in the input procedure.
   section-name-2 identifies the last section in the input procedure. It need only be specified when the input procedure consists of more than one section.

5. section-name-3 identifies the first or only section in the output procedure.
   section-name-4 identifies the last section in the output procedure. It need only be specified when the output procedure consists of more than one section.

6. file-name-1, file-name-2,..., file-name-3, file-name-4,... must be defined in a file description (FD) entry in the Data Division.

7. file-name-1, file-name-2,..., file-name-3, file-name-4,... must be

   a) sequential files.

   b) The size of each logical data record described for these files must be equal to the size of each logical data record described for the sort-file. When the data descriptions of the elements which make up these records are not identical, the user must describe these records in such a way that they occupy the same number of character positions in main memory.

8. At least one ASCENDING/DESCENDING phrase must be specified in a SORT statement.

9. The SORT statement may be written anywhere in the program except

   a) in the declaratives area,

   b) in an input/output procedure belonging to a SORT statement.

Programming considerations

1. The collating sequence is set by means of the ASCENDING/DESCENDING option:

   a) When ASCENDING is specified, sorting proceeds from the lowest to the highest value of the sort-key, i.e. in ascending order.

   b) When DESCENDING is specified, sorting proceeds from the highest to the lowest value of the sort-key, i.e. in descending order.

   The collating sequence is governed by the same rules as apply to the comparison of operands in "Relation Conditions".

2. If DUPLICATES is specified and several data records have the same contents in all of their sort items, then the return sequence of these records is as follows:

   a) If no input procedure is specified, the records are returned in the order in which the associated files were specified in the SORT statement. If records with identical sort item contents exist in one and the same file, the records are returned in the order in which they were entered.

   b) If an input procedure is specified, the records are returned in the order in which they left the input procedure.

3. If DUPLICATES is not specified and several data records have the same contents in all of their items, then the return sequence of these records is undefined.

4. When the program is executed, the collating sequence for the comparison of nonnumeric sort items is set as follows:

   a) If COLLATING SEQUENCE has been specified in the SORT statement, this entry is used as a sort criterion.

   b) If COLLATING SEQUENCE was not specified in the SORT statement, the program-specific collating sequence will be used (see "OBJECT-COMPUTER Paragraph").

5. INPUT PROCEDURE indicates that the Procedure Division contains an input procedure to process records prior to sorting. If INPUT PROCEDURE is specified, control passes to it when the input division of the SORT program is ready to accept the first record. During RELEASE statement processing the input procedure releases records to the sort-file (see "RELEASE Statement"). The user may not leave the input procedure. The compiler inserts a return mechanism at the end of the last section in the input procedure, i.e. once the last statement in the input procedure has been processed the input procedure is terminated and the released records will be sorted in the sort-file. The following rules apply to the input procedure, which is a self-contained section within the Procedure Division:

   a) It must consist of one or more sections that are written consecutively and that do not form part of an output procedure.

   b) It must contain at least one RELEASE statement so that records can be released to the sort-file (see "RELEASE Statement").

   c) It may run only in conjunction with an associated SORT statement.

d) It may include any procedures needed to select, modify, or copy the copy the records, with the following restrictions:

- The input procedure must not contain a SORT statement.

- It is forbidden to branch from within an input procedure to procedure-names outside that procedure.
  However, the compiler permits the ALTER, GO TO and PERFORM statements to reference procedure-names outside the input procedure. The programmer must make sure that a transfer of this type is followed by a return to the input procedure in order to effect a proper exit from this procedure (i.e. to process its last statement).

- It is forbidden to branch from points outside an input procedure to procedure-names within that procedure.
  However, the compiler permits this provided the branch does not involve a RELEASE statement or the end of the input procedure.

6. An input procedure, when specified, is processed before the records in the sort-file are sorted.

7. The USING phrase is specified when there is no input procedure. File-name-1, file-name-2,... is the name of the input file to be sorted. All records contained therein are automatically released to the file indicated by sort-file-name. Thus, file-name-1, file-name-2,... may not be opened while the SORT statement is executing.

   The SORT statement opens, processes, and closes this input file automatically.

8. If the programmer specifies error handling and/or label processing declaratives for file-name-1, file-name-2,... the SORT statement will make the necessary linkage to the appropriate Declaratives Section.

9. OUTPUT PROCEDURE means that the Procedure Division contains an output procedure in which records are processed after sorting. When the OUTPUT PROCEDURE phrase is used, control is passed to the output procedure after the sort-file has been processed by the SORT command. During RETURN statement processing the output procedure accepts the data records from the sort-file. The user may not leave the output procedure. The compiler inserts a return mechanism at the end of the last section in the output procedure, i.e. once the last statement in the output procedure has been executed the procedure is terminated and control passes to the statement that follows the SORT statement.

   The following rules apply to the output procedure, which is a self-contained section within the Procedure Division:

   a) It must consist of one or more sections that are written consecutively and that do not form part of an input procedure.

   b) It must contain at least one RETURN statement to make the sorted records available for processing.

   c) It may run only in conjunction with an associated SORT statement.

   d) It may include any procedures needed to select, modify, or copy the records as they are released.

Restrictions

- The output procedure must not contain a SORT statement.

- It is forbidden to branch explicitly within an output procedure to procedure-names outside that procedure.
  However, the compiler permits the ALTER, GO TO and PERFORM statements in the output procedure to reference procedure-names outside that procedure. The programmer must make sure that a transfer of this type is followed by a return to the output procedure in order to effect a proper exit from this procedure (i.e. to process its last statement).

- It is forbidden to branch from points outside an output procedure to procedure-names within that procedure.
  However, the compiler permits this provided the branch does not involve a RETURN statement or the end of the output procedure.

10. An output procedure, when specified, is processed after the records in the sort-file have been sorted.

11. When the INPUT PROCEDURE or OUTPUT PROCEDURE phrase is used, a branch is performed in the program as if it were a format-1 PERFORM statement. This means that all sections that form the procedure are run once and procedure execution terminates once the last statement has been processed. Thus, either procedure (or both) may be terminated by an EXIT statement.

12. When SORT statements appear in segmented programs, the following restrictions apply:

    a) If a SORT statement appears in a section that is not located in an independent segment, then all input or output procedures referred to by that SORT statement must be:

       - either wholly contained within one fixed segment,

       - or wholly contained in a single independent segment.

    b) If a SORT statement appears in an independent segment, then all input or output procedures referred to by that SORT statement must be:

       - either wholly contained within one fixed segment,

       - or wholly contained in the same independent segment as the SORT statement.

    These restrictions do not apply to the compiler described in this manual.

13. The GIVING phrase is specified when there is no output procedure. File-name-3, file-name-4,... is the name of the output file. All the records from the file indicated by sort-file-name are automatically released to this output file. Thus, file-name-3, file-name-4,... may not be opened while the SORT statement is executing. The SORT statement opens, processes and closes this output file automatically.

14. If the programmer specifies error handling and/or label processing declaratives for file-name-3, file-name-4,... the SORT statement will make the necessary linkage to the appropriate Declaratives Section.

15. Since the SORT statement is not directed at individual records, it does not conform to the standard input/output statements (READ, WRITE, etc.). The READ statement, when executing, reads a single record; likewise, the WRITE statement writes an individual record. The SORT statement, on the other hand, does not treat an individual record but an entire file. Thus, this entire file must be placed at the disposal of SORT, either via the USING phrase or by repeated use of the RELEASE statement within an input procedure, before SORT can function. The SORT routine alters the sequence of the records within the file, and hence the first record returned by the SORT routine is, as a rule, not the first record released to the routine. SORT cannot provide any output before it has received the whole of the input.

10-3 SPECIAL REGISTERS FOR SORT

In the compiler, four special registers are provided for communication between the programmer and the SORT control routine. Each of these registers is a 4-byte binary data item with a fixed name; the description of each register is PICTURE S9(5) USAGE IS COMPUTATIONAL. The data description entries for the registers are generated automatically by the compiler and cannot be declared by the programmer.

1. SORT-FILE-SIZE register

The programmer may set the contents of the SORT-FILE-SIZE register to the approximate number of records to be processed in the next sort. This must be done before the SORT statement is executed. From this information, the sort routine calculates how much space it will need on internal working files. SORT-FILE-SIZE is initially set to zero by the compiler; and, if the value of the register is still zero at the time a sort begins, the SORT control routine will make a default space allocation. Thus, the programmer does not have to enter any estimate of file size into this special register, although supplying such information aids the efficiency of the sort. The value entered in the special register by the user (e.g. MOVE 25 TO SORT-FILE-SIZE) is released to the control routine.

2. SORT-CORE-SIZE register

The programmer may set the contents of the SORT-CORE-SIZE register to the number of bytes of memory that are to be available for use by the SORT routine. This must be done before the SORT statement is executed.

3. SORT-MODE-SIZE register

SORT-MODE-SIZE may be set if variable-length records are to be sorted.
The programmer may set this register to the most commonly used record length in the input file. The appropriate value must be moved into SORT-MODE-SIZE before the execution of the SORT statement. The compiler initializes the contents of this register to zero; and, if the programmer has not supplied any other value before the SORT statement is executed, the maximum record length will be assumed to be the most common record length.

4. SORT-RETURN register

After a SORT RELEASE/RETURN statement has been executed, the special register SORT-RETURN contains a value to indicate whether the sort has been successful. A value of zero indicates that the sort has been successful. A non-zero value indicates that the sort terminated abnormally.

5. Summary

None of the special registers (except SORT-RETURN) has its value reset or set to zero by the execution of the SORT statement. If a program contains several sorts, the programmer must move appropriate values into the SORT-FILE-SIZE, SORT-CORE-SIZE and SORT-MODE-SIZE registers before each sort is executed.

It should be noted that the special registers are binary items and, therefore, cannot be used as immediate operands of ACCEPT statements.

Example

```
ACCEPT MODE-SIZE FROM TERMINAL.
MOVE MODE-SIZE TO SORT-MODE-SIZE.
```

Since the ACCEPT statement cannot be used to transfer information directly to the SORT-MODE-SIZE register, the MOVE statement is used for this purpose.

10

EXAMPLES

<u>Example 1</u>

Sort processing with an output file

```
ID DIVISION.
PROGRAM-ID. SORT1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER-FILE ASSIGN TO DA-DISC-S-SYS011.
    SELECT MASTER-FILE ASSIGN TO DA-DISC-S-SYS012.
    SELECT SORT-FILE   ASSIGN TO DISC.
DATA DIVISION.
FILE SECTION.
FD  MASTER-FILE LABEL RECORD STANDARD.
01  MASTER-RECORD.
    02  E0              PIC X.
    02  E1              PIC 9(4).
    02  E2              PIC 9(4).
    02  E3              PIC 9(4).
FD  OUTPUT-FILE LABEL RECORD STANDARD.
01  OUTPUT-RECORD.
    02  A0              PIC X.
    01  A1              PIC 9(4).
    02  A2              PIC 9(4).
    02  A3              PIC 9(4).
SD  SORT-FILE LABEL RECORD STANDARD.
01  SORT-RECORD
    02  S0              PIC X.
    02  S1              PIC 9(4).
    02  S3              PIC 9(4).
    02  S4              PIC 9(4).
PROCEDURE DIVISION.
    SORT SORT-FILE ASCENDING S1 S2 S3                 ]
                                                      }-------------------- (01)
         USING MASTER-FILE GIVING OUTPUT-FILE.        ]
    STOP RUN.
```

(01) The sort operation takes place in the following stages:

1. The records are released from the input file MASTER-FILE to the SORT-FILE.

2. The records are sorted in the sort-file according to ascending S1, or (in those records with identical S1) according to ascending S2, or (in records with identical S1 and S2) according to ascending S3.

3. The records are released from the sort-file to the OUTPUT-FILE.

S2 -> ?

Example 2

Sort processing with two output files

```
ID DIVISION.
PROGRAM-ID. SORT2.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER-FILE   ASSIGN TO DA-DISC-S-SYS011.
    SELECT OUTPUT-FILE-1 ASSIGN TO DA-DISC-S-SYS012.
    SELECT OUTPUT-FILE-2 ASSIGN TO DA-DISC-S-SYS013.
    SELECT SORT-FILE     ASSIGN TO DISC.
DATA DIVISION.
FILE SECTION.
FD  MASTER-FILE LABEL RECORD STANDARD.
01  INPUT-RECORD.
    02  E0          PIC X.
    02  E1          PIC 9(4).
    02  E2          PIC 9(4).
    02  E3          PIC 9(4).
FD  OUTPUT-FILE-1 LABEL RECORD STANDARD.
01  OUTPUT-RECORD-1 PIC X(13).
FD  OUTPUT-FILE-2 LABEL RECORD STANDARD.
01  OUTPUT-RECORD-2 PIC X(13).
SD  SORT-FILE LABEL RECORD STANDARD.
01  SORT-RECORD.
    02  S0          PIC X.
    02  S1          PIC 9(4).
    02  S2          PIC 9(4).
    02  S3          PIC 9(4).
PROCEDURE DIVISION.
MAIN SECTION.
M1.
    OPEN INPUT MASTER-FILE
         OUTPUT OUTPUT-FILE-1 OUTPUT-FILE-2.
    SORT SORT-FILE ASCENDING S1 S2 S3
                                                    (01)
         INPUT PROCEDURE IPROC
         OUTPUT PROCEDURE OPROC.
    CLOSE MASTER-FILE OUTPUT-FILE-1 OUTPUT-FILE-2.
ME.
    STOP RUN.
IPROC SECTION.
IPO.
    READ MASTER-FILE AT END GO TO EE.
    IF E0 NOT = "C" RELEASE SORT-RECORD             (02)
                FROM MASTER-RECORD.
    GO TO IPO.
IPE.
    EXIT.
OPROC SECTION.
OPO.
    RETURN SORT-FILE AT END GO TO OPE.
    IF S0 = "A" WRITE OUTPUT-RECORD-1 FROM SORT-RECORD   (03)
          ELSE WRITE OUTPUT-RECORD-2 FROM SORT-RECORD
    GO OPO.
OPE.
    EXIT.
```

Sort processing

(01) 1. Control passes to the input procedure (IPROC SECTION).
2. The records in the sort-file are sorted in ascending order according to S1 S2 S3.
3. Control passes to the output procedure (OPROC SECTION).

(02) A record is read from the input file. Only those records without a "C" in their first character position are to be processed. If the INPUT-RECORD is valid, it is released to the sort-file as SORT-RECORD. This process continues until the end of the input-file is encountered. Control then returns to the SORT statement.

(03) A record is released from the sort-file. If its first character position contains an "A", it is written to the first output file (OUTPUT1). All other records are written to the second output file (OUTPUT2). When all the records in the sort-file have been processed, the statement following the SORT statement is executed.

Example 3

```
ID DIVISION.
PROGRAM-ID. MERGE1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER-FILE-1 ASSIGN TO DA-DISC-S-SYS011.
    SELECT MASTER-FILE-2 ASSIGN TO DA-DISC-S-SYS012.
    SELECT OUTPUT-FILE-1 ASSIGN TO DA-DISC-S-SYS013.
    SELECT OUTPUT-FILE-2 ASSIGN TO DA-DISC-S-SYS014.
    SELECT SORT-FILE ASSIGN TO DISC.
DATA DIVISION.
FILE SECTION.
FD  MASTER-FILE-1 LABEL RECORD STANDARD.
01  MASTER-RECORD-1.
    02  E10            PIC X.
    02  E11            PIC 9(4).
    02  E12            PIC 9(4).
    02  E13            PIC 9(4).
FD  MASTER-FILE-2 LABEL RECORD STANDARD.
01  MASTER-RECORD-2.
    02  E20            PIC X.
    02  E21            PIC 9(4).
    02  E22            PIC 9(4).
    02  E23            PIC 9(4).
FD  OUTPUT-FILE-1 LABEL RECORD STANDARD.
01  OUTPUT-RECORD-1 PIC X(13).
FD  OUTPUT-FILE-2 LABEL RECORD STANDARD.
01  OUTPUT-RECORD-2 LABEL RECORD STANDARD.
SD  SORT-FILE LABEL RECORD STANDARD.
01  SORT-RECORD.
    02  S0             PIC X.
    02  S1             PIC 9(4).
    02  S2             PIC 9(4).
    02  S3             PIC 9(4).
PROCEDURE DIVISION.
MAIN SECTION.
M01.
    MERGE SORT-FILE ON ASCENDING S1 S2 S3 ]
    USING INPUT-FILE-1 INPUT-FILE-2       }  ---------------  (01)
    GIVING OUTPUT-FILE-1 OUTPUT-FILE-2.   ]
M02.
    STOP RUN.
```

(01) The records from two files are sorted according to the same criteria and output to two identical files in sorted order.

All files are sorted in ascending order according to the sort keys S1 S2 S3.

Example 4

```
ID DIVISION.
PROGRAM-ID. MERGE2.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER-FILE-1 ASSIGN TO DA-DISC-S-SYS011.
    SELECT MASTER-FILE-2 ASSIGN TO DA-DISC-S-SYS012.
    SELECT OUTPUT-FILE-1 ASSIGN TO DA-DISC-S-SYS013.
    SELECT OUTPUT-FILE-2 ASSIGN TO DA-DISC-S-SYS014.
    SELECT SORT-FILE     ASSIGN TO DISC.
DATA DIVISION.
FILE SECTION.
FD  MASTER-FILE-1 LABEL RECORD STANDARD.
01  MASTER-RECORD-1.
    02  E10            PIC X.
    02  E11            PIC 9(4).
    02  E12            PIC 9(4).
    02  E13            PIC 9(4).
FD  MASTER-FILE-2 LABEL RECORD STANDARD.
01  MASTER-RECORD-2.
    02  E20            PIC X.
    02  E21            PIC 9(4).
    02  E22            PIC 9(4).
    02  E23            PIC 9(4).
FD  OUTPUT-FILE-1 LABEL RECORD STANDARD.
01  OUTPUT-RECORD-1 PIC X(13).
FD  OUTPUT-FILE-2 LABEL RECORD STANDARD.
01  OUTPUT-RECORD-2 PIC X(13).
SD  SORT-FILE LABEL RECORD STANDARD.
01  SORT-RECORD:
    02  S0             PIC X.
    02  S1             PIC 9(4).
    02  S2             PIC 9(4).
    02  S3             PIC 9(4).
PROCEDURE DIVISION.
MAIN SECTION.
M01.
    OPEN OUTPUT OUTPUT-FILE-1 OUTPUT-FILE-2.
M02.
    MERGE SORT-FILE ON ASCENDING S1 S2 S3 ]
    USING MASTER-FILE-1 MASTER-FILE-2     } ------------------ (01)
    OUTPUT PROCEDURE IS OPROC1.           ]
M03.
    CLOSE OUTPUT-FILE-1 OUTPUT-FILE-2.
    STOP RUN.
OPROC SECTION.
OP01.
    RETURN SORT-FILE AT END GO OP02.        ]
    WRITE OUTPUT-RECORD-1 FROM SORT-RECORD.  } --------------- (02)
    WRITE OUTPUT-RECORD-2 FROM SORT-RECORD. ]
OP02.
    EXIT.
```

(01) The records of both files are sorted according to the same criteria and released to the sort-file in sorted order. Control then passes to the output procedure (OPROC SECTION).

(02) The sort-file is released record-by-record and written to the output files.

11 DEBUGGING AIDS

11-1 GENERAL DESCRIPTION

For testing, the COBOL language provides two standard language elements: Debugging lines and the WITH DEBUGGING MODE clause.

11.2 ENVIRONMENT DIVISION LANGUAGE ELEMENTS

DEBUGGING LINE

Function

Debugging lines are COBOL source program lines to be used for testing. After the COBOL source program has finished debugging, these lines may be kept unchanged in the program; if the programmer instructs the compiler to treat these lines as comments by omitting the WITH DEBUGGING MODE clause from the SOURCE-COMPUTER paragraph.

Format

A "D" in the indicator field (column 7) of the source program line defines this source program line as a debugging line.

Rules

The contents of a debugging line must be such that the compilation of the source program produces a syntactically valid program irrespective of any WITH DEBUGGING MODE clause specified in the SOURCE-COMPUTER paragraph.

Several successive debugging lines are permitted. Debugging lines may be continued; however, the continuation line, too, must contain a D in the indicator field. It is not permitted to split character strings between several lines.

Debugging lines, when used, must not precede the OBJECT-COMPUTER paragraph in the source program.

The presence or absence of the WITH DEBUGGING MODE clause in the SOURCE-COMPUTER paragraph decides whether the compiler treats all debugging lines of the program as normal statement lines or comment lines.

WITH DEBUGGING MODE CLAUSE

Function

The WITH DEBUGGING MODE clause specifies that all debugging lines are to be included in the compilation as normal source program lines.

Format

WITH <u>DEBUGGING</u> <u>MODE</u>

Rules

1. This clause is part of the SOURCE-COMPUTER paragraph.

2. If the WITH DEBUGGING MODE clause is specified, all debugging lines (that is, all those lines which have a D in column 7, the indicator field) will be compiled.

3. If the WITH DEBUGGING MODE clause is omitted, then all debugging lines will be treated as comment lines by the compiler.

12 LIBRARIES

12.1 COPY STATEMENT

Function

The COPY statement incorporates text from a library into a source program.

Format

```
COPY text-name [ {OF | IN} library name ] [SUPPRESS]

[ REPLACING word-1 BY {word-2 | literal-1 | identifier-1} [ word-3 BY {word-4 | literal-2 | identifier-2} ] ... ].
```

Rules

1. The word COPY must be preceded by a space. Each COPY statement must end with a period.

2. text-name consists of 1 to 30 characters. It may only contain letters, digits and hyphens.

   It must begin with an alphabetic character.

   The hyphen must not be specified in the eight position.

   Of text-name, only the first eight characters are used by the system. Every text-name in a library must therefore be unique from the first through the eight position.

3. library-name is the name of a library file that may contain several texts with different names. It must be a valid link-name.

4. The SUPPRESS phrase causes logging of COPY text in the source listing to be suppressed (see "COB1 User's Guide" [7]).

5. word-1, word-2, etc., can be any COBOL word except COPY.

6. A COPY statement can appear anywhere in a COBOL source program where a COBOL word or a separator is allowed. It must not appear inside another COPY statement.

7. text-name is the name of the text to be copied into the source program. The texts are stored in the library under this name.

8. If texts from various libraries are to be copied into a source program, text-name must be qualified with the link-name of the library.

9. If copying is to take place from a single LMS source program or an LMS program library, the program or library can be given the link name COBLIB (see COB1 User's Guide" [7]).

10. When a COPY statement is executed, the library-text associated with text-name is copied into the source program and the entire COPY statement is logically replaced, beginning with the word COPY and ending with the period.

11. The REPLACING phrase causes every word-1, word-3, etc., in the copy to be replaced by whatever is specified after BY.

12. The original text in the library is not changed by the REPLACING phrase.

13. A copy member is compiled as though it were actually part of the source program.

14. If the end of the named library text is encountered, the copy operation terminates for a given COPY statement.

Programming considerations

1. A COPY text must not contain any further COPY statements.

2. The syntactic correctness of the library text cannot be determined independently of the source program. It is impossible to check the correctness of the entire source program until all COPY statements have been executed in their entirety.

3. The library text must conform to the rules of the COBOL program image format (see chapter 1, "COBOL Coding Form").

LITERATURE

[1] BS1000
BS2000
TRANSDATA PDN
System Standards
Reference Manual

Target group
Users of Siemens mainframes
Contents
Operating system standards for BS1000, BS2000 and TRANSDATA PDN, standards for data values; codes for character representation.

[2] **BS2000**
Utility Routines
Reference Manual

Target group
BS2000 users (non-privileged)
Contents
Utility routines for non-privileged BS2000 users.
Applications
BS2000 timesharing mode

[3] **UDS** (BS2000)
Application Programming
User's Guide

Target group
Programmers
Contents
Transaction concept, use of the currency table,
COBOL-DML, CALL-DML
Testing of DML-functions

[4] BS2000
DMS Tape Processing
Reference Manual

Target group
BS2000 users, assembly language programmers (both non-privileged)
Contents
Functions of the Data Management System in BS2000;
DMS commands and macros, service and action macros;
access methods UPAM, SAM and BTAM for tape files
Applications
BS2000 interactive/batch mode, programming

[5] **SINIX-PC-MX/X**
Operating Guide
ANIMATOR

Target group
LEVEL II COBOL programmers
Contents
Description of LEVEL II COBOL
ANIMATOR debugging tool

[6] BS2000
Control System Command Language
Reference Manual

Target group
BS2000 users (non-privileged)
Contents
All BS2000 system commands in alphabetical order with detailed explanations and examples.
The following products are dealt with:
BS2000-GA, MSCF, JV, FT, TIAM
Applications
BS2000 interactive/batch mode, procedures

[7] BS2000
COB1
COBOL Compiler
User's Guide

Target group
COBOL users in BS2000
Contents
Operating of the COB1 compiler and the software required for the development, linking and debugging of COBOL programs; structures of the COB1 system and the generated object modules; programming hints; compiler messages and COB1 interface with UDS.

[8] BS1000
BS2000
TRANSDATA PDN
Systems Standards
Reference Manual

Target group
Users of Siemens mainframes
Contents
Operating system standards for BS1000, BS2000 and TRANSDATA PDN; standards for data volumes, codes for character representation

[9] **COB1** (BS2000)
COBOL Compiler
Reference Guide

Target group
COBOL users in BS2000
Contents
Tabular representation of all language elements of the COB1 COBOL compiler. Operation of the compiler by means of BS2000 commands and COBRUN statements.

[10] AID (BS2000)
Advanced Interactive Debugger
Reference Manual

Target group
First-time AID users
Contents
Performance characteristics and applications; operation of AID for error diagnosis, error correction and debugging.

[11] **AID**
Advanced Interactive Debugger
Debugging at machine code level
User's Guide

<u>Target group</u>
Programmers in BS2000
<u>Contents</u>
Descriptions of all AID commands that are available for debugging at machine code level.
Messages.
<u>Applications</u>
Debugging of programs in interactive/batch mode.

[12] AID (BS2000)
Advanced Interactive Debugger
Debugging of COBOL Programs
User's Guide

<u>Target group</u>
COBOL programmers.
<u>Contents</u>
Preparations for the symbolic testing of COBOL programs.
Description of all AID commands that are available for symbolic testing.
Example of an AID session.
Messages.
<u>Applications</u>
Debugging of COBOL programs in batch/interactive mode.

From

Name . . . .

Company . . . .

Street . . . .

City/Postal Code . . . .

Phone . . . .

Date . . . .

To
Siemens AG

K D ST QM 2
Manualredaktion

Otto-Hahn-Ring 6
D-8000 München 83

Reader's Reply Form

COB1 (BS2000)
COBOL Compiler

Reference Manual Part 2
Edition August 1986 (Software Product COB1 V2.3A)

Order No. U996-J-Z55-6-7600

Page	Criticisms/Suggestions/Corrections

Page	Criticisms/Suggestions/Corrections

From

Name ....

Company ....

Street ....

City/Postal Code ....

Phone ....

Date ....

To
Siemens AG

K D ST QM 2
Manualredaktion

Otto-Hahn-Ring 6
D-8000 München 83

Reader's Reply Form

COB1 (BS2000)
COBOL Compiler

Reference Manual Part 2
Edition August 1986 (Software Product COB1 V2.3A)

Order No. U996-J-Z55-6-7600

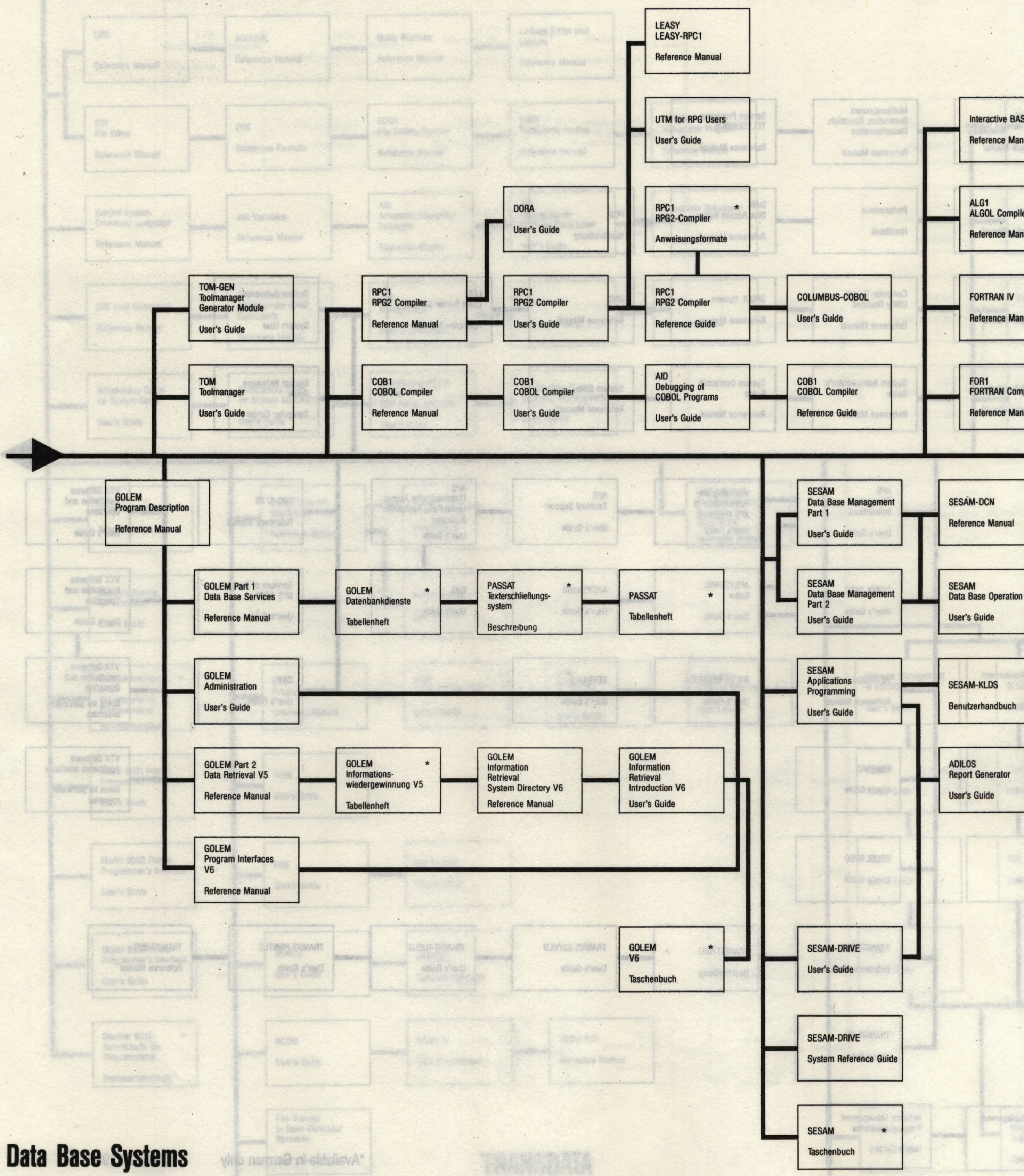
Page	Criticisms/Suggestions/Corrections

Page	Criticisms/Suggestions/Corrections

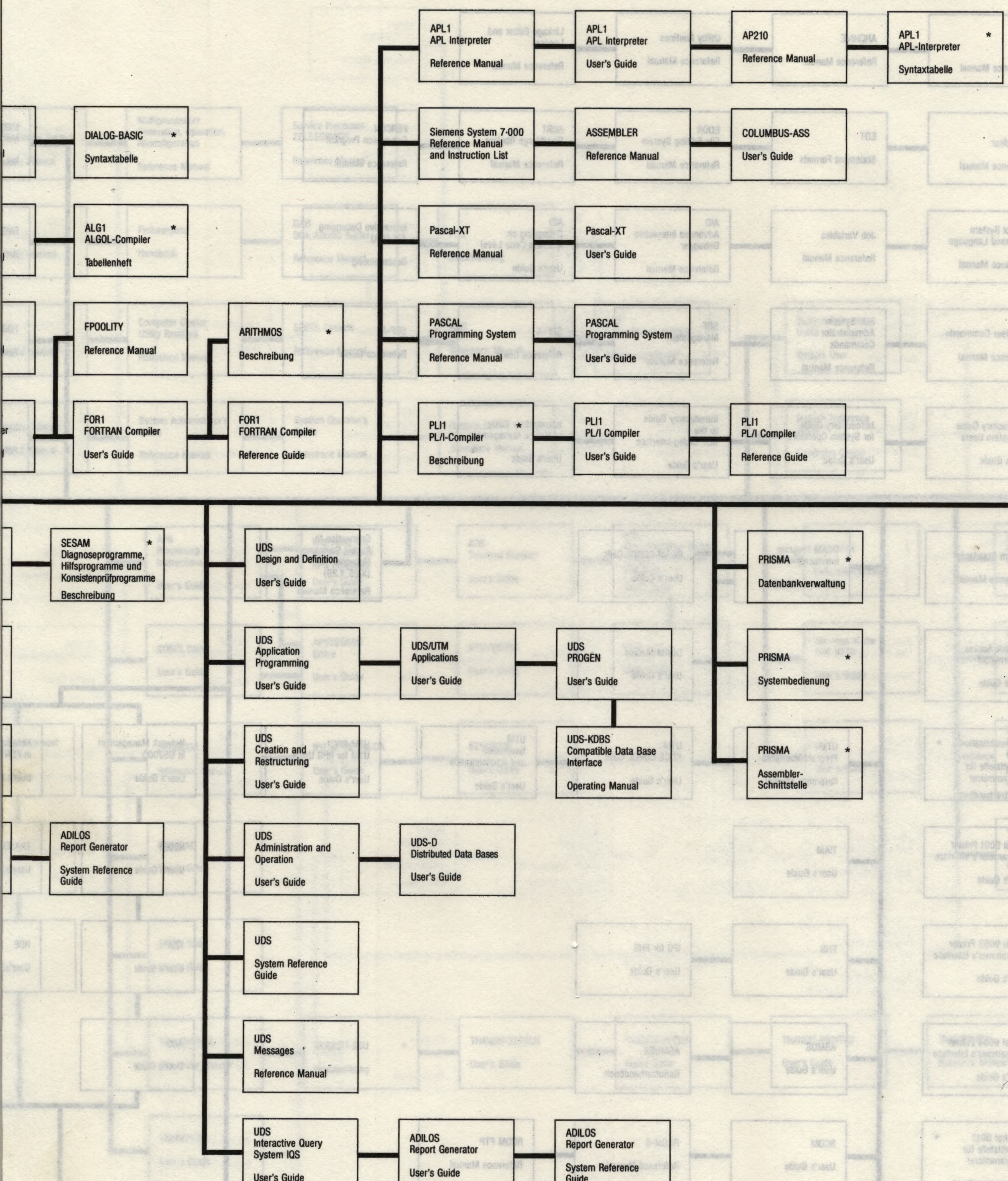
Published by Bereich Datentechnik
Postfach 83 09 51, D-8000 München 83

Siemens Aktiengesellschaft

Programming Systems



Data Base Systems

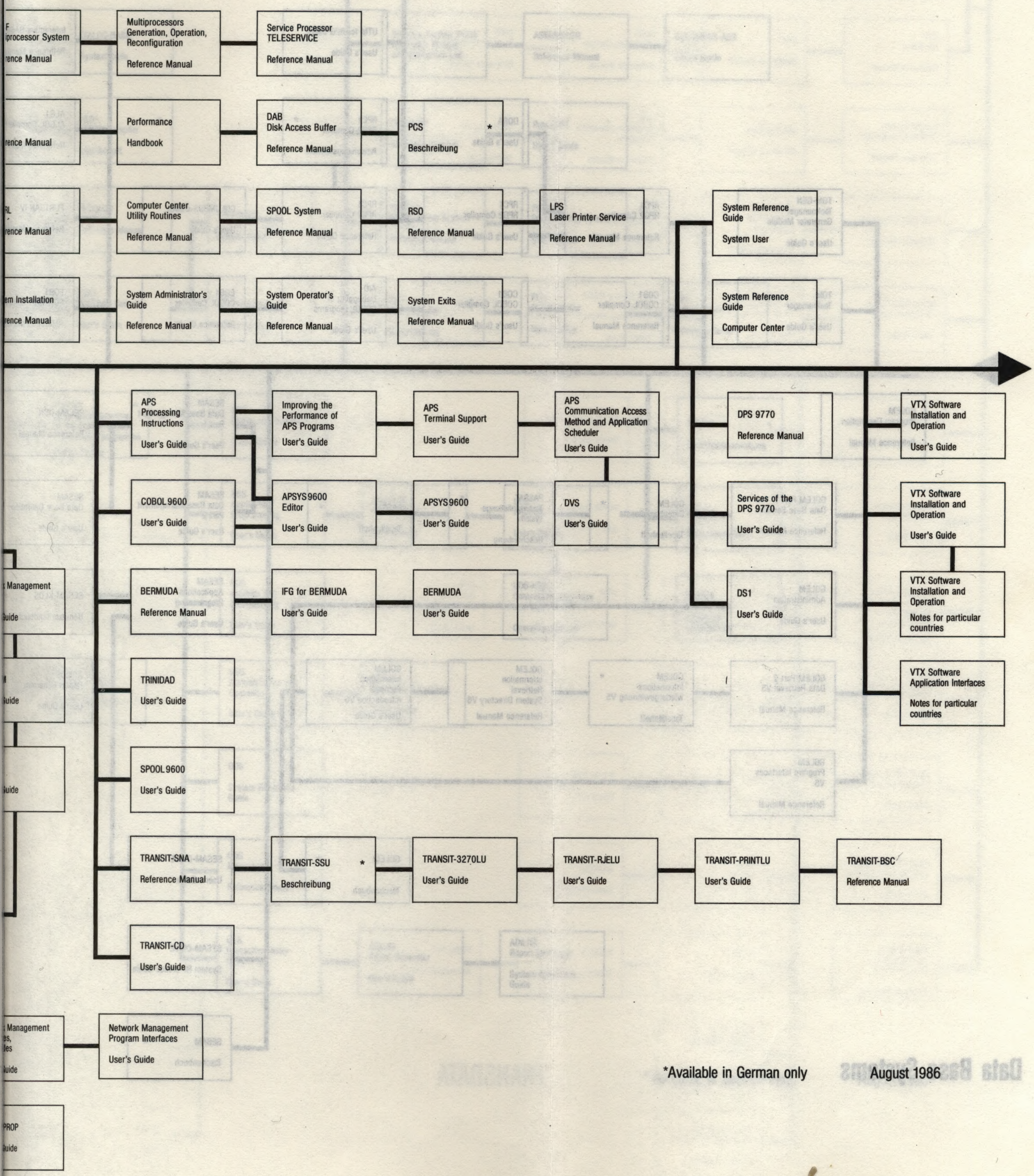


* Available in German only.

August 1986

Publications Plan for

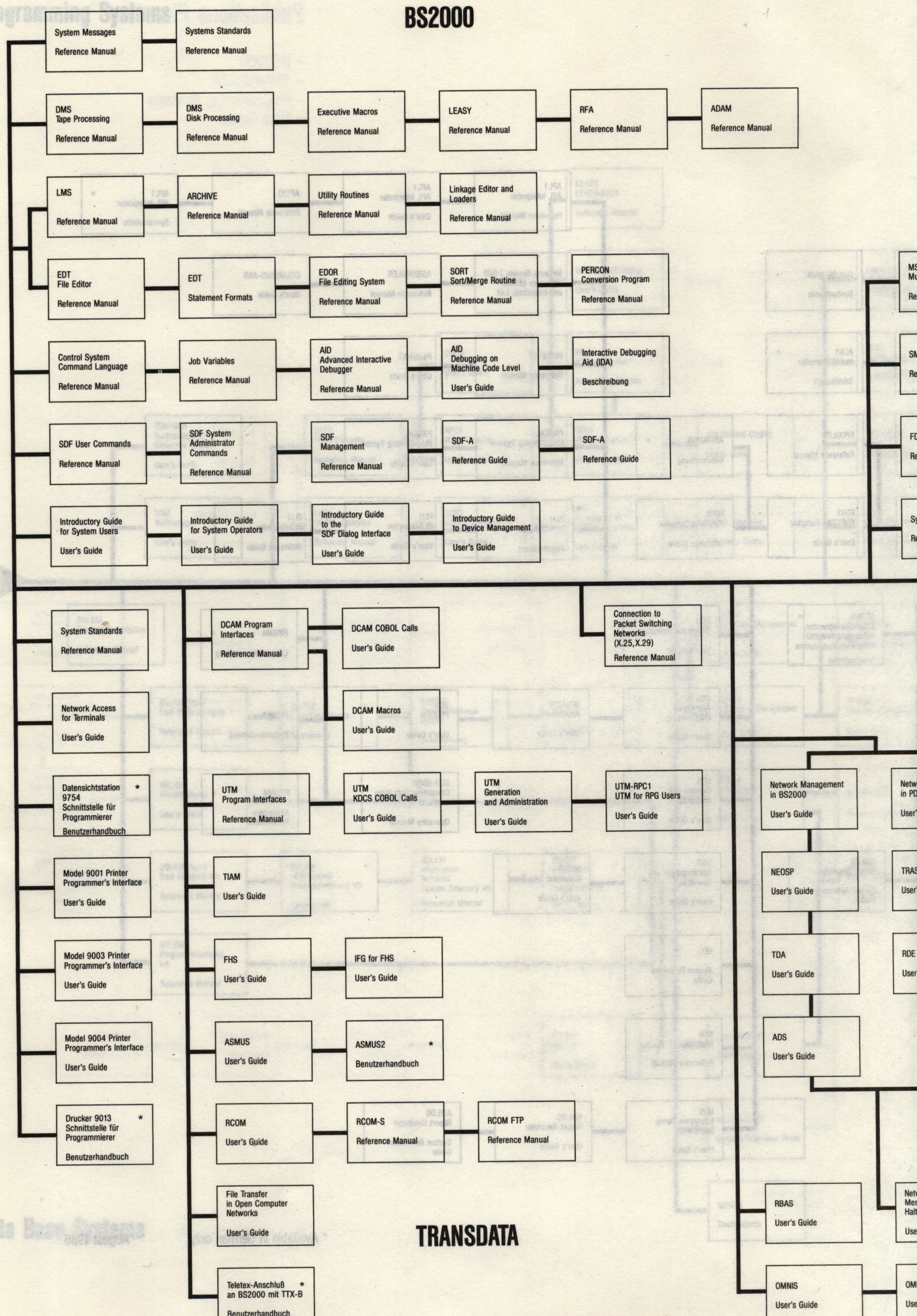
- BS2000
- TRANSDATA
- Programming Systems
- Data Bases



*Available in German only

August 1986

BS2000



TRANSDATA